VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hudba na drátě
Music on-line

2012 Bc. Tomáš Buriánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Buriánek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Hudba na drátě
Music on-line

Zásady pro vypracování:

Cílem práce je vytvořit program podporující výuku hry na různé hudební nástroje (například kytara nebo flétna) a vytváření hudebních stop pomocí notového zápisu.
Software bude umožňovat:
1. Vytvářet notové zápisy neomezené délky pro jeden hudební nástroj.
2. Skládat notové zápisy dohromady a vytvářet tak zápis pro několik hudebních nástrojů.
3. Přehrávat notové zápisy s možností vypínání jednotlivých hudebních nástrojů.
4. Notový zápis bude možno zobrazit v několika náhledech, jako je třeba zápis tabů nebo akordů pro kytaru.
5. Vytvářet alternativní náhledy na notový záznam pro určitý nástroj. Jedná se o alternativní zobrazení noty nebo jejich kombinace, které pak bude použito k nahrazení notového zápisu.
6. Podporu výuky formou zobrazování aktuálně hraného tónu (noty) v notovém zápisu s možností dalších podpůrných nástrojů jako je metronom nebo ladička.
7. Export a import do standardních formátů.

Práce bude obsahovat:
1. Zhodnocení podpůrných rámců pro vývoj Java desktopových aplikací, jako jsou například javadesktop.org, JavaFX a další.
2. Implementaci softwaru.
3. Dokumentaci a popis struktury softwaru.

Seznam doporučené odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
Oracle [online]. 2010 [cit. 2011-06-04]. Java Sound API. Dostupné z WWW: <http://www.oracle.com/technetwork/java/index-jsp-140234.html>.

Dále podle pokynů vedoucího práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: 2. května 2012 …….

Rád bych poděkoval Ing. Davidovi Ježkovi Ph.D za odbornou pomoc a konzultaci při vytváření této diplomové práce a Tomáši Rulíškovi za rady ohledně hudební teorie.

Abstrakt

Hlavní náplní této práce je tvorba programu, který bude sloužit pro vytváření hudebních stop pomocí notového zápisu a také dalších alternativních zápisů. Hudební stopy je pak možné skladádat dohromady a tvořit tak komplexnější hudební skladbu. Program vhodně slouží pro výuku hry na různé nástroje jako je například kytara. Dále umožňuje ukládání hudby do standartních formátů. Také obsahuje další doplňkové nástroje pro výuku, jako jsou kytarová ladička a meotronom. Práce obsahuje popis základní hudební teorie, včetně notového zápisu, potřebné pro tvorbu aplikace a také popis dalších alternativnich zápisů not. Dále je nastíněno zhodnocení podpůrných rámců pro vývoj Java desktopových aplikací, jako jsou například javadesktop.org a JavaFX. Práce obsahuje dokumentaci a popis struktury programu.

Klíčová slova

Java Standard Edition, JavaFX, JDesktop, Swing, Notová osnova, Nota, Tón, Tabulatura, MIDI.

Abstract

The main purpose of this work is to create a program that will serve to create music tracks with musical notation as well as other alternative staves. Music tracks is composed together to form the complex musical composition. The program is also suitably used for teaching to play on various instruments such as guitar. It allows exporing of music into standard formats. It also contains another tools for teaching, such as guitar tuner and metronome. The work contains a description of basic music theory, including musical notation, necessary for creating program and a description of other alternative forms of notes. There are described support frameworks for developing Java desktop applications, such as javadesktop.org and JavaFX. The work contains the documentation and description of program structure.

Key words

Java Standard Edition, JavaFX, JDesktop, Swing, Note stave, Note, Tone, Tabulature, MIDI

Obsah

1 Úvod

Hudba je pevně spjata s člověkem již od počátků historie. Tak jako věda a umění se i hudba a pohled na ni vyvíjel. Vznikaly nové hudební nástroje, pomocí kterých bylo možno hudbu vytvářet. S tím se i hudebníci snažili nějakým způsobem zachytit hudbu do hmatatelné podoby. Pomocí různých značek zapisovali hudební tóny na papír a tvořili tak první noty. Časem dostaly tóny svá jména, zápis not se sjednotil a vznikl tak dnes již známý notový zápis, pomocí kterého je možné zachytit hudební skladbu zápisem not do notové osnovy. Vývoj se však zdaleka nezastavil. S příchodem elektroniky bylo najednou možno reprodukovat hudbu pomocí hudebních aparatur, vznikly elektrické kytary, ale také další hudební pomůcky jako elektronické ladičky, elektronické metronomy, hudbu bylo možné zaznamenat na pásek či gramofonovou desku. I v počítačovém věku má hudba své místo. Počítače se využívají nejen pro poslech hudby, ale také pro tvorbu hudby a výuku. Vznikají tak nové programy pro střih, či tvorbu hudebních skladeb pomocí notové osnovy, kde tužku a papír vystřídal počítač s myší a klávesnicí. Vznikají také nové alternativní zápisy not, které jsou většinou odvozeny od hry na určitý nástroj. Díky internetu se setkáváme s hudbou každý den. Hudební skladby a zápisy skladeb např. notovou osnovou jsou tak dostupnější. S tím však je spjata i tématika ochrany autorských práv.

Mým cílem bylo vytvořit program, pomocí kterého by bylo možné zapisovat hudbu pomocí notového zápisu. Dále aby bylo možné vhodnou formou podporovat učení se hry na nějaký jako je například kytara či jiný nástroj.

V kapitole **2** je nastíněna základní hudební teorie, která byla nutná při návrhu a implementace aplikace. Jde například o vlastnosti tónů a způsob zápisu not do notové osnovy. Kapitola **2.6** se pak zabývá dalšími alternativním notovými zápisy.

Pro tvorbu aplikace bude využito vývojové platformy Java. Proto jsou v kapitole **3** popsány podpůrné rámce, které je možné využívat pro vývoj Java desktopových aplikací a jejich zhodnocení. Jsou zde nastíněny technologie, které je možné využívat přímo v rámci Java Standard Edition, ale také další technologie, které byly vyvinuty pro doplnění funkcí v rámci různých projektů. V kapitole **3.2** je pak popsána nová Java platforma pro vývoj tzv. Rich Client Aplikací – JavaFX.

Kapitola **4** se zabývá přímo implementací programu. Je zde rozebrána celková struktura programu a jsou zde popsány všechny důležité části. Kapitola **4.4** se konkrétně zabývá notovou hudební stopou, do které je možné pomocí notového zápisu zapisovat noty. Využitím architektury Model-View-Controller je možné na zápis not pohlížet i z několika různých alternativních pohledů. V kapitole **4.5**

je doplněna další hudební stopa a to bicí, zaznamenávající údery bicích nástrojů. Kapitola **4.6** se zabývá implementací simulace virtuálních nástrojů, mezi které patří piáno, kytara nebo baskytara. V kapitole **4.7** jsou uvedeny doplňkové podpůrné nástroje aplikace. Je to ladička hudebních nástrojů a metronom. Export a import hudebních dat je popsán v kapitole **4.9** .

Kapitola **5** se zabývá implementací doplňkové aplikace ve vývojové platformě JavaFX. Jde o obdobnou simulaci virtuálního hudebního nástroje piáno jako v hlavním programu. Nachází se zde také porovnání s vývojem aplikace pomocí Java SE a komponent Swing.

2 Hudební Teorie

Pro porozumění problému je nutné znát několik základních informací o tom, jak pracovat například s notovými osnovami a dalšími prvky ze základů hudební nauky. Následující informace o hudební nauce byly čerpány ze zdroje : **[1]**

2.1 Zvuk a tóny

Zvuky jsou vše, co slyšíme. Vznikají chvěním hmoty, tj. chvěním různých těles, kapalin, plynů. Zpěv ptáků, hluk motorů, údery bubnů, troubení, pískání – to vše jsou zvuky. Chvějící se těleso rozechvívá okolní vzduch a ten rozechvívá sluchové ústrojí v našem uchu; toto chvění v uchu vnímáme jako zvuk. Zvuky dělíme na tóny a hluky. Některá tělesa kmitají stále stejně rychle, chvějí se pravidelně, jiná kmitají nestejně rychle, nepravidelně. Tóny vznikají pravidelným chvěním, hluky nepravidelným chvěním. Tóny mají určitou přesnou výšku, kterou můžeme napodobit zpěvem nebo na hudebním nástroji. Hluky jsou všechny ostatní zvuky kromě tónů, například rány, šumy, praskání aj. Mají výšku neurčitou. Sice můžeme určit jestli znějí vysoko či hluboko, ale nelze jejich výšku stanovit tak přesně jako u tónu. V hudbě užíváme především tónů. Jsou to tóny hudebních nástrojů a lidského hlasu (zpěvu). Kromě tónů se však v hudbě uplatňují některé hluky. Příkladem mohou být bicí a perkusivní nástroje, triangl, činely aj.

2.2 Vlastnosti tónů

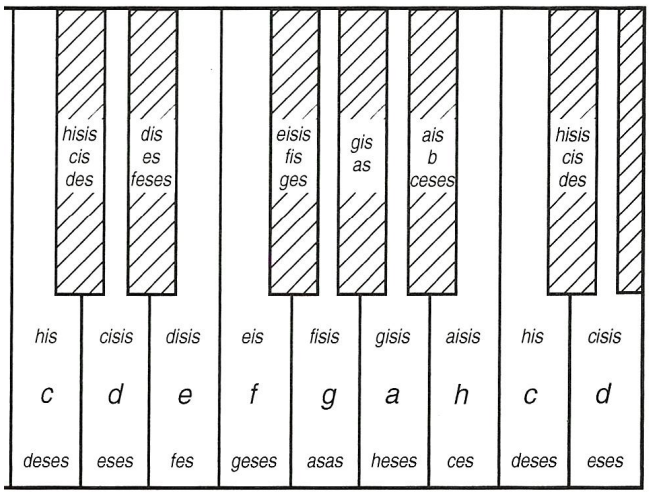
Základní vlastnosti tónů jsou čtyři: *délka*, *síla*, *barva, výška*. Tyto termíny se běžně používají v hudební nauce. Podle toho jak dlouho tóny znějí, rozlišujeme jejich různou délku. Podle toho jak silně znějí rozlišujeme různou sílu. Sílu bychom mohli označit také jako hlasitost tónu. Tóny se liší od sebe i různou barvou. Různé barvy tónu označujeme především podle původu tónů, tedy například tóny hudebních nástrojů – klavír, kytara, housle, tóny mužských, ženských hlasů, tóny klaksonu apod. Podle výšky rozlišujeme tóny hluboké a vysoké. Výšku tónu určujeme přesně počtem kmitů tělesa za vteřinu – frekvence. Nejhlubší tóny mají kmitočet kolem 20 kmitů, vysoké tóny několik tisíc kmitů za vteřinu. Kmitočet tónů střední polohy s pohybuje kolem 500 kmitů za vteřinu.

2.3 Tónová soustava a jména tónů

Tónová soustava je přehledné uspořádání všech tónů, užívaných v hudbě, podle jejich výšek. Nevšímá si délek, ani barvy či síly tónů. Základem naší tónové soustavy je sedm tónů, které se jmenují *c, d, e, f, g, a, h*. Těchto sedm tónů se v tónové soustavě opakuje několikrát v různých výškových polohách. Souhrnně se nazývají základní tónová řada. Od výchozího tónu *c* k nejbližšímu opakovanému *c* nacházíme osm stupňů, které se souhrnně nazývá oktáva. Tónová soustava obsahuje celkem devět oktáv, z nichž každá má své jméno, například: subkontra oktáva, kontra, velká, malá, jednočárkovaná až pětičárkovaná. Podobně se nazývají i jednotlivé tóny v oktávě.

2.4 Tóny zvýšené a snížené

Každý tón naší tónové soustavy můžeme jednou nebo dvakrát zvýšit nebo snížit. Jednoduché zvýšení označujeme v názvu tónu příponou *–is*, dvojité zvýšení *–isis*. Jednoduché snížení označujeme *–es*, dvojité *–eses*. Tóny c, d, e, f, g, a, h se považují za *základní*. Tóny zvýšené a snížené se souhrnně nazývají tóny odvozené nebo alterované. Základních a odvozených tónů je v rámci oktávy celkem 35. V oktávě však rozlišujeme pouze dvanáct různých tónových výšek. Z toho vyplývá, že dva až tři tóny různého jména mají stejnou výšku, neboli že tóny stejné výšky se mohou jmenovat různě. Na *Obrázek 2.1* je znázorněna klaviatura, na které jsou vypsány názvy všech základních a odvozených tónů.



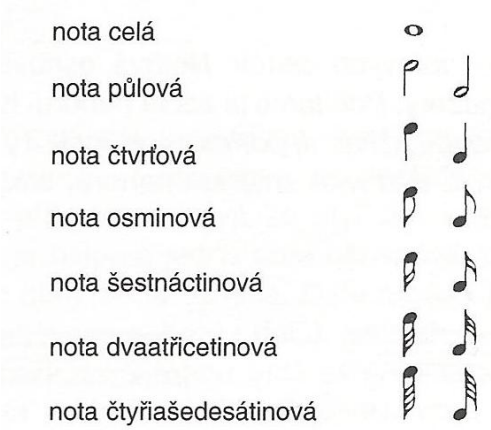
Obrázek 2.1: Obrázek klaviatury s vyznačenými tóny

Klaviaturu má například klavír, akordeon, nebo varhany. Klaviatura se skládá z bílých a černých kláves. V rozmezí jedné oktávy je vždy 12 kláves, sedm bílých a 5 černých. Základní tóny *c*, *d*, *e*, *f*, *g*,

a, h jsou umístěny pouze na bílých klávesách. Černé klávesy se po celé klaviatuře střídají vždy dvě a tři. Tón c je vždy umístěn na bílé klávese vedle nižší (levé) ze dvou černých kláves a podle něho můžeme snadno určit umístění tónů ostatních. Nejmenší vzdálenost ve výšce dvou tónů užívaná v naší hudbě se nazývá půltón a nachází se vždy mezi dvěma sousedními klávesami.

2.5 Notový zápis

Tóny zapisujeme notami. Délku tónu označujeme tvarem noty a výšku tónu umístěním noty v notové osnově. Barva tónu je dána zvoleným obsazením, k označení síly tónů, rychlosti hry a způsobu přednesu slouží zvláštní označení. Používané tvary not jsou zobrazeny na *Obrázek 2.2*.



Obrázek 2.2: Obrázek základních not notového zápisu

Jak napovídají již tyto názvy, nota celá se rovná délkou dvěma půlovým, čtyřem čtvrťovým, osmi osminovým atd. Nejbližší menší nota má vždy hodnotu poloviční a vyšší dvojnásobnou. Podle toho nota čtvrťová se rovná čtyřem šestnáctinovým. Noty mají tyto části: hlavičku(vyplněnou či nevyplněnou) nožku, praporec (jednoduchý, dvojitý, trojitý nebo čtyřnásobný).

Noty píšeme do notových osnov. Notová osnova má pět linek a mezi nimi čtyři mezery. Počítáme je zdola nahoru. Kromě toho pod osnovou i nad osnovou užíváme pomocných linek. Noty se píší na linky a do mezer, na pomocné linky, nad i pod pomocné linky.

Noty které se nacházejí nad sebou v jednom úseku tvoří souzvuk či více hlas. Na kytaře souzvuk také nazýváme akord. Na obrázku *Obrázek 2.3* je vidět příklad zápisu not do notové osnovy.

Určení jména noty v osnově určuje znaménko zvané klíč. Píše se vždy na začátku notové osnovy na každém řádku. Klíč udává umístění jedné noty a podle ní se řídí pojmenování not ostatních. Základní

Obrázek 2.3: Ukázka zápisu různých not do notové osnovy

tónová řada pak má své postavení v každém klíči jiné. Nejčastěji používané klíče jsou *G* klíč (houslový) určuje postavení noty *g1* na druhé lince a *F* klíč (basový) určuje postavení noty *f* na čtvrté lince.

Odvozené (alterované) tóny se v notovém písmu značí posuvkami. Jednoduché zvýšení označuje křížek #, dvojité zvýšení dvojkřížkem. Jednoduché snížení se značí bé *b*, dvojité snížení dvojité bé *bb*.

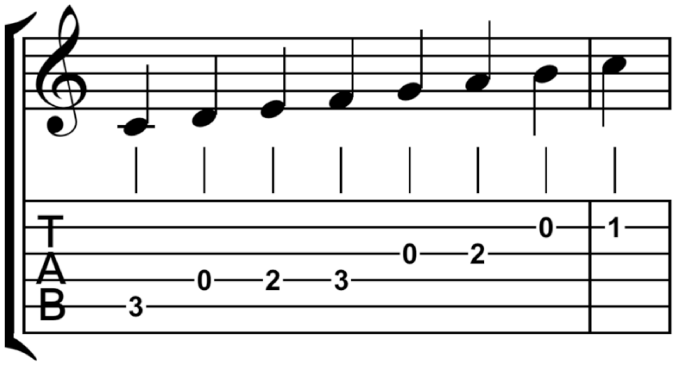
2.6 Alternativní notové zápisy

V průběhu historie vývoje notového zápisu, který měl několik různých podob se vyvíjely i různé alternativní zápisy, pomocí kterých bylo také možné zaznamenat hudební skladbu. Většinou jde o hudební notace, které zaznamenávají nástrojový prstoklad, či jiné značení, které je typické pro hru na konkrétní nástroj. Pro hru na nástroj je pak jednodušší si představit, jak zahrát požadovaný tón, na rozdíl od notové osnovy, kde je výšku tóna určena pouze pozicí v osnově. Není tedy potřeba znalosti notopisu, ani umístění konkrétních tónů na nástroji. Souhrnně se tyto alternativní zápisy nazývají *tabulatury*. Existuje mnoho tabulatur. Nejvíce se však používají pro pražcové strunné nástroje, tedy kytarové nástroje. Tabulatura je známá již od Renesance a Baroka. V minulém století však tabulaturu zasáhla obroda díky popularizaci kytarových hudebních nástrojů jako jsou klasická akustická kytara, elektrická kytara, či baskytara, a to ve všech možných hudebních stylech.

2.6.1 Kytarová tabulatura

V porovnání se standardním notopisem je tabulatura blíže visuální reprezentaci nástrojového hmatníku. Proto je jednodušší se naučit pracovat s tímto zápisem. Nevyžaduje tak zjišťování pozice tónu na hmatníku. Ta je zaznamenána přímo v zápise, na rozdíl od notového zápisu. Na obrázku *Obrázek 2.4* , který byl získán ze zdroje **[2]**, je zobrazena ukázka srovnání tabulaturového kytarového zápisu s notovým zápisem.

Tabulaturová osnova se tedy skládá z vodorovných linek. Jednotlivé linky představují strunu kytarového nástroje. Jednotlivé struny kytary jsou pak seřazeny od nejvyšší strunu po nejhlubší strunu odshora dolu. Přibližuje tak pohled hráče na kytaru při hře. Na linkách jsou pak umístěna čísla představující taby. Jeden tab představuje ekvivalent k jedné notě z notové osnovy. Číslo tabu určuje pražec, na kterém je příslušná struna držena. Dále je někdy uvedeno



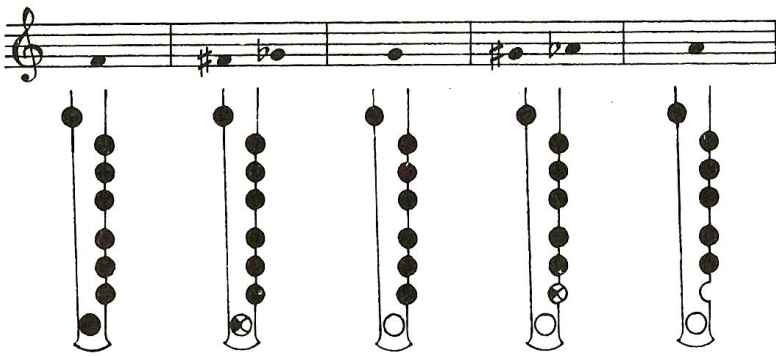
Obrázek 2.4: Ukázka srovnání tabulaturového kytarového zápisu s notovým zápisem

vlevo na začátku každé linky písmeno určující ladění příslušné struny, pokud například není ladění nástroje uvedeno dříve, nebo se jedná o nestandardní ladění nástroje. Taby se zapisují postupně zleva doprava, obdobně jako v notovém zápise. Taby které se nacházejí nad sebou tvoří souzvuk tónů, pro kytaru též akord. Pro délku tónu nelze použít u tabů tvar noty, jako je tomu u notové osnovy, což je nevýhoda. Délka tónu může být zaznamenána například písmenkem v prostoru nad osnovou, které přísluší k tabům umístěným pod ním. Písmenko pak určuje, jaká je to nota. Většinou jsou použita písmenka z anglických názvů not: W – whole (celá), H – half (půlová), Q – quarter (čtvrťová) atd. Ve většině tabulaturových zápisech skladeb, které lze nalézt na internetu se však nepoužívá žádné zaznamenání délky tónu. Tím se ztrácí velmi důležitá informace, protože bez poslechu, nebo znalosti zaznamenané skladby se pak těžce dá odvodit, který tón má jakou délku a bylo by možno tuto skladbu reprodukovat jiným způsobem, než jak je tomu v originálním podání. Většinou se tato ztráta informace neřeší, protože se hráč většinou učí skladbu s jejím hudebním záznamem. Nejlepším způsobem je připojení ekvivalentního notového zápisu, ve kterém jsou délky tónů zaznamenány tvarem noty.

Výhodou je, že tabulaturu je možné zapsat díky tzv. *ASCII tab*, což je textový souborový formát, pomocí kterého je možné tabulaturu reprezentovat jako čísla, písmena a znaky ASCII. Z toho důvodu je tak rozšířený na internetu, a to díky jednoduché přenositelnosti. Soubory mohou mít příponu *.tab*, *.btab* anebo pouze prosté *.txt*. S rozšiřitelností na internetu je však spojena otázka autorských práv na zaznamenanou skladbu. Existují různé webové portály, které poskytují kytarové tabulatury, které jsou převážně tvořeny jinými uživateli. Použití těchto tabů je pak možné jen pro soukromé účely, vlastní učení, nikoliv však pro veřejnou reprodukci. Jsou však i specializované portály, které nabízejí placené profesionální verze tabulatur.

2.6.2 Další tabulatury

Existují i další různé alternativní zápisy not. Velké využití mají například zápisy dechových nástrojů, jako je flétna, kde se zakresluje pro hru nějakého tónu ucpání dírek nástroje. Na obrázku *Obrázek 2.5* získaného ze zdroje **[3]** je zobrazeno srovnání notového zápisu s tabulaturou flétny.



Obrázek 2.5: Porovnání notového zápisu s tabulaturou flétny

Dále existují například tabulatury pro bicí nástroje, které zaznamenávají údery do jednotlivých bubínků či činelů.

3 Zhodnocení podpůrných rámců pro vývoj Java desktopových aplikací

V dnešní době s přicházejícími novými úkoly a problémy jsou kladeny nové a stále složitější nároky na všechny vývojové platformy, které se zabývají desktopovými aplikacemi. To neminulo ani samotnou Java platformu, pro kterou jsou nové nároky nelehkým úkolem, hlavně z důvodu její nezávislosti na operačním systému. Vzniklo tak mnoho projektů zaměřených na vývoj a podporu desktopových aplikací, některé zanikly, avšak mnoho jich přineslo řadu nových technologií, které tak bylo možno integrovat do Java Standard Edition a je možné je plnohodnotně využívat při vývoji aplikací. Poslední novinkou společnosti Sun Microsystems je platforma JavaFX, která přináší nová řešení pro vytváření tzv. RIA (Rich Internet Application) – bohatých internetových aplikací. Lze ji tak použít pro vývoj nejen webových, ale také desktopových či mobilních aplikací. V následujících kapitolách bude zobrazen přehled a zhodnocení technologií, které je možné v rámci Java platformy využít.

3.1 Java Standard Edition

Mnoho desktopovcýh Java technologií jsou většinou integrovány přímo v *Java Standard Edition (Java SE)*. Další jsou dostupné odděleně například v rámci open source projektů apod. a je možné je doinstalovat. Technologie, které lze v rámci Java SE používat jsou následující:

- **Java Web Start/ JNLP** technologie poskytuje pružné a robustní vývojové řešení pro Java aplikace. Poskytuje architekturu, pomocí které je možné aplikace závislé na webovém prohlížeči přenést pomocí protokolu JNLP na klientský desktop.
- **Swing -** aplikační rozhranní (API) poskytuje komplexní balík komponent pro grafické uživatelské rozhranní, které umožňují vývoj kvalitních desktopových aplikací. Swing staví na dalších desktopových technologiích jako jsou JavaBeans, AWT, Java2D atd.
- **Java 2D** je balík tříd pro pokročilejší 2D grafiku a práci s obrázky. Zahrnuje práci s tvary, linkami, textem, obrázky, obsahuje rozšířenou podporu pro kompozici a nastavení alfa kanálu.
- **Java Sound** API poskytuje podporu pro práci se zvukem a audiem. Například přehrávání hudby, zaznamenávání, mixování, práce s MIDI jako sekvencování a syntetizace zvuku. Je tak možné vhodně vylepšit aplikaci.

- **AWT (Abstract Windows Toolkit)** API podporuje programování pro grafické uživatelské rozhranní. Je jádrem Java SE knihoven. Obsahuje robustní událostní model. Dále třídy pro grafiku a práci s obrázky. Důležitými jsou také Layout Managery pro flexibilní rozložení grafických prvků v oknech. Také obsahuje základní balík komponent uživatelského rozhranní jako jsou okna, tlačítka atd. AWT tvoří základ pro Swing, který na něm stojí. Mnoho komponent ze Swing je doporučeno pro práci namísto obdobných komponent z AWT. Obsaženo v Java SE.

V současné době je pro tvorbu grafického uživatelského rozhranní Java desktopových aplikací nejvíce využíváno technologie **Swing**. Podrobnosti o jejích funkcích a komponentách, které poskytuje budou nastíněny v následující kapitole **3.1.1** .

Lze také doinstalovat další technologie, které je možné v rámci Java SE používat. Příkladem je třeba **Java 3D** API, které umožňuje vytvářet a renderovat 3D objekty a scénu. Ostaní doinstalovatelné technologije je možné vidět ve zdroji **[4]**.

Pro desktopové Java aplikace se rozvinula řada projektů, které se zabývají rozšířením a vývojem nových technologií pro Java platformu. Mnoho z nich rozšiřuje a vytváří nové komponenty založené na Swing komponentách, další se snaží o grafické zlepšení a podporu tvorby grafického uživatelského rozhranní (GUI) aplikací, jiné projekty zase usilují o zvýšení nativní podpory. Na komunitním webu **java.net**, odkaz v referenci **[5]**, který se zabývá Java technologiemi se nachází přehled mnoha různých projektů, které se zabývají právě Java desktopovými aplikacemi. Je možné zde zjistit informace o konkrétním projektu, zobrazit repositář zdrojových kódů projektu a případně se přesměrovat na webové stránky projektu. Největším projektem je projekt **Swinglabs**, který navrhuje různá rozšíření především pro Java Swing komponenty. Cílem je vytvářet a rozšiřovat GUI funkcionality, které jsou požadovány od tzv. Rich Client Aplikací. Tento projekt se dělí na mnoho pod-projektů. V kapitolách **3.1.2** , **3.1.3** a **3.1.4** budou popsány nejvýznamnější pod-projekty, které nabízí nová řešení a významně ovlivnily samotné Java SE.

3.1.1 Swing

Informace o této technologii byly čerpány ze zdrojů **[6]** a **[7]**. Swing je součástí *Java Foundation Classes (JFC)*, což je API zahrnující celkovou práci s grafickým uživatelským rozhranním (GUI) Java programů. Byl vyvinut tak, aby nabídnul balík vylepšencýh grafických komponent, než je tomu u *Abstract Window Toolkit (AWT)* API. Swing poskytuje přirozený vzhled a design. Může

emulovat vzhled mnoha různých platforem operačních systémů a umožňuje nastavit i vlastní vzhled nezávisle na této platformě. Obsahuje mnoho užitečných komponent, které většinou nahrazují komponenty z AWT. Příkladem mohou být komponenty `JButton`, `JCheckBox` apod. Nabízí i nové komponenty, které poskytují rozšířené funkce a možnosti jako je například `JTabbedPane`. Na rozdíl od AWT komponent nejsou Swing komponenty implementovány kódem specifickým pro platformu, ale jsou psány výhradně v Javě, což poskytuje platformní nezávislost. Výhodou je, že Java komponenty je možné rozšířit, a to o vlastní implementací použitím dědícího mechanismu. Pro vykreslování se používá vlastní Java 2D API namísto nativních nástrojů, což zvyšuje jeho flexibilitu. Velmi často Swing komponenty využívají návrhového vzoru Model-View-Controller. To znamená, že odděluje data, která jsou v komponentě zobrazována od uživatelského rozhranní, pomocí kterého se data zobrazují. Díky tomu je možno přiřadit mnoha Swing komponentám modely dat. Je možné nastavit základní, již implementované modely, kterým se pouze data předají, nebo je možné pomocí příslušných rozhranní vytvořit model vlastní. Typickými komponentami, které očekávají nějaký model dat jsou například `JTable`, `JComboBox`, nebo `JList`. S tím jsou i spjaty události, které jsou vyvolávány nad komponentami, které je možné následně poslouchat a pracovat tak s modely.

Pro rozložení komponent v GUI se nejvíce využívá tzv. Layout Managerů z AWT. Je možné využít předdefinovaných, jako jsou například `BorderLayout` a dalších. Komponentu je pak možné vložit do jiné komponenty, která slouží jako kontejner zobrazující více potomků. Je možné pomocí constraint objektu specifikovat omezení a rozložení vkládané komponenty do kontejneru. Kontejnery mohou tvořit například JPanel a jiné komponenty dědící ze třídy Container. Další možností je použít vlastního layoutavacího mechanismu a to implementováním rozhranní **`LayoutManger`** nějaké třídě. V této třídě je pak nutné implementovat příslušné metody. Ty například definují velikost rodičovského kontejneru. Nejdůležitější metodou je však metoda **`layoutContainer`**. V ní by se mělo nastavovat rozložení, tedy pozice a velikost komponent obsažených v kontejneru na základě jejich vlastností jako preferovaná velikost apod. Nastavení se provádí například metodami **`setSize`** a **`setLocation`** nebo pomocí **`setBounds`**.

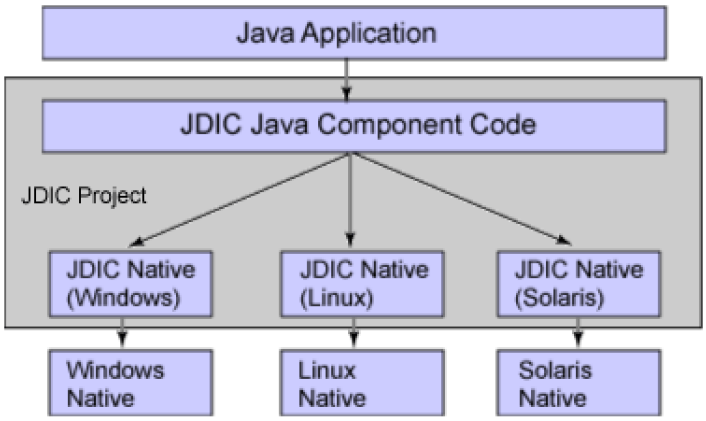
Změnu vzhledu komponent je možné provést buďto děděním existujících komponent a změnou vykreslování a nebo změnou jiných vlastností. Další možností je použití tzv. „Look and Feel“ vzhledu. Každá komponenta má definován svůj vlastní vzhled. Vzhledy všech komponent jsou definovány v objektu **`LookAndFeel`**. Díky třídě **`UIManger`** lze tento vzhled metodou **`setLookAndFeel`** změnit na jiný. UIManger poskytuje sadu vzhledů, mezi které patří ať už klasický vzhled Windows, či defaultní Metal vzhled a další. Je možné také nastavit vzhled pomocí javax.swing.plaf.synth třídě,

která umožňuje definovat vzhled pomocí XML souboru. Díky této třídě byl definován nový vzhled **`Nimbus`**, který je vykreslován díky Java 2D vektorové grafice, namísto statických bitmap, jak je tomu u základních stylů. To znamená že může být vykreslován v jakémkoliv rozlišení a je dále vysoce nastavitelný.

3.1.2 JDesktop Integration Components

Informace ohledně projektu JDesktop Integration Components byly čerpány ze zdrojů **[8]**, **[9]** a **[10]**. Tento projekt byl vytvořen za účelem vyvinout nové komponenty, které by poskytovaly Java aplikacím přístup ke službám operačního systému, tak jako nativní aplikace. (Nativní aplikace – aplikace napsaná v programovacím jazyce, který používá vestavěné knihovny operačního systému, jakou jsou např. Visual C++ aplikace, které běží na Windows). Problémem bylo, že Swing a celkově Java nepodporuje u svých aplikací ty samé vlastnosti jako nativní aplikace. Důvodem je to, že Java aplikace nemohou být ve skutečnosti plně nativní, protože Java kód je psán filozofií, aby aplikace běžela na jakémkoliv operačním systému. Nativní funkce se pak volají pomocí Java Native Interface. Cílem bylo to, aby nové komponenty podporovaly volání nativních funkcí na kterémkoliv OS a zároveň si zachovaly svoji platformní nezávislost. JDIC tak vytváří nativní přístupy k jednotlivým platformám a obaluje je do jednoho JDIC API. Na obrázku *Obrázek 3.1* je naznačena struktura JDIC API.

Pro správnou funkci aplikací, které využívají JDIC je však nutné poskytnout knihovny operačního systému, které budou Java třídy volat. Pro každý operační systém je pak nutná jiná knihovna. Je nutné mít tedy tři základní věci: **jdic.jar** – je to soubor s Java třídami potřebnými pro vývoj JDIC aplikace, **jdic.dll** a **tray.dll** – jsou soubory přemosťující metody z jdic.jar a nativní metody OS.



Obrázek 3.1: Struktura a propojení JDIC API

JDIC poskytuje komponenty jako je například **`WebBrowser`**, která doplňuje mezery v zobrazování webových stránek. Umožňuje totiž, oproti Swing komponentě JEditorPane, která umí zobrazit obsah webu pomocí HTML kódu, navíc zobrazit i rozšířené funkce webových stránek, jako jsou různé skripty apod. Ke správné funkci totiž využívá vestavěných prohlížečů jako jsou Internet Explorer, nebo Mozilla tak jako ostatní nativní aplikace. Umožňuje tak zobrazovat samotný obsah webových stránek, kromě ovládacích prvků, které však není již těžké naimplementovat vlastní, které využívají pouze funkcí komponenty.

Další zajímavou komponentou je **`SystemTray`**, která umožňuje Java aplikaci umístit svoji ikonu **`TrayIcon`** do tzv. *system tray*, což je například pro OS Windows pravá část hlavního panelu, kde se nacházejí další ikony různých aplikací. Je možné této ikoně předat mnoho funkcí pro obohacení funkcionality aplikace.

Další komponenta **`Desktop`** slouží k použití standardních aplikací pro otevření souborů. Například soubor s příponou **.doc** může být z Java aplikace otevřen pomocí asociovaného programu Microsoft Word. Příklady funkcí, které třída Desktop řeší:

- `Desktop.mail(Message m)`: Automaticky otevře systémem preferovaný e-mailový program s vyplněným poli.
- `Desktop.browse(URL url)`: otevře webovou stránku v systémovém prohlížeči
- `Desktop.edit(File f)`: Použije program asociovaný s příkazem edit pro otevření souboru
- `Desktop.open(File f)`: použije program asociovaný se souborovým type s příkazem open pro otevření souboru
- `Desktop.print(File f)`: vytiskne soubor

JDIC dále poskytuje další komponenty, například **`FileTypes`**, pomocí které je možné asociovat přímo z Java aplikace soubory s jinými aplikacemi.

JDIC projekt měl překlenou mezery, které měla Java z hlediska nativní podpory, což se mu podařilo. Tento projekt se však již dále příliš nevyvíjí. Hlavním úkolem byla ale integrace těchto komponent do Java Standard Edition, což se povedlo v roce 2006 do Java SE 6. Více se dá dočíst zde : **[11]**

Do Java SE 6 se podařilo integrovat podporu **SystemTray** a **Desktop**, popsané výše. Lze je nalézt `java.awt.*` API. Těchto komponent bylo využito při vytváření vlastní aplikace.

3.1.3 Swing Application Framework

Dalším projektem z dílny **SwingLabs** je specifikace, která by měla poskytovat jednoduchý aplikační rámec určený pro práci se Swing komponentami a rozšíření o další komponenty. Informace byly čerpány ze zdroje **[12]** a **[13]**. Swing Appliaction Framework definuje infrastrukturu, která je blízká běžným desktopovým aplikacím. To by mělo usnadnit práci při vytváření nové aplikace. Podporuje správu životního cyklu od spuštění po ukončení celé aplikace. Dále usnadňuje přístup a načítání lokálních zdrojů. Tento framework je součástí vývojového prostředí **NetBeans** ve verzi 7.0. Při vytváření nového projektu je možné si vybrat novou aplikaci **Java Desktop Application**. Tím se vytvoří skeleton desktopové aplikace založené na Swing Application Framework (JSR 296), kterou tvoří základní prvky aplikace jako je hlavní rám aplikace, panel menu, nebo panel status bar.

Samotná aplikace je rozdělena na dvě části a to *aplikační* – třída dědící ze **`SingleFrameApplication`** sloužící pro inicializaci aplikace a spuštění a *pohledová* – tvoří vzhled aplikace, třída dědící z **`FrameView`**. Samotné NetBeans IDE poskytuje návrhový nástroj, pomocí kterého se zobrazí grafický náhled na uživatelské rozhranní aplikace a díky tomu je možné dále upravovat vzhled aplikace. To se provádí například vkládáním a upravováním grafických komponent jako jsou tlačítka, textová pole apod. ze Swing, AWT, Java Beans a dalších z Java SE API. Tyto komponenty je možné různě řadit, přiřazovat jim akce, které budou provádět, nastavovat vlastnosti atd. Manipulací s grafickým náhledem se automaticky generuje kód nastavení pozic a vlastností ve zdrojovém kódu pohledu `FrameView`. Mezi náhledy na kód a na design je možné přepínat tlačítky **Source** a **Design**. Vygenerovaný kód lze najít v regionu **`Generated Code`**, který by se neměl manuálně měnit, aby byla zachována správná funkce aplikace. Nastavení rozložení komponent se pak provádí pomocí objektů ze třídy **`GroupLayout`**.

Novou funkcionalitu přináší třída **`TaskMonitor`**. Tato třída může sloužit jako model různým GUI komponentám, jako jsou status bary apod. sloužící k zobrazení stavu úkolů, které aplikace provádí v pozadí. Také poskytuje přehled všech úkolů v rámci aplikačního kontextu, tak i stav jednoduchého úkolu v popředí. Je možné vytvořit vlastní úkol zděděním třídy **`Task`**, ve kterém se provádí funkce úkolu. Ten je pak možné provádět na pozadí a pomocí TaskMonitor zjišťovat jeho stav provedení a zobrazovat jej v grafickém rozhranní.

Další užitečnou třídou je **`ResourceMap`**. Ta usnadňuje získávání zdrojů aplikace jako jsou různé hodnoty, textové řetězce nacházející se například v titulcích tlačítek, oken aplikace atd. nebo obrázkové ikony, které je možné využít pro vylepšení grafického vzhledu aplikace.

Bohužel vývoj tohoto frameworku byl od roku 2009 zastaven. Sice byl dále distribuován v rámci NetBeans, ale ve verzi NetBeans 7.1 již není součástí. Swing Application Framework knihovnu lze však i dále využívat pro tvorbu desktopových aplikací.

3.1.4 Další projekty v rámci Swinglabs

V rámci Swinglabs vznikla další řada mnoha zajímavých projektů. Další informace tomto projektu se lze dozvědět zde: **[14]** a zde: **[15]** Některé nové komponenty, kterými se zabývaly tyto projekty, jsou již také obsaženy v Java SE, konkrétně od verze 6. Mezi zajímavé projekty patří:

- **SwingX** rozšiřuje funkcionality swing komponent.
- **Nimbus** - zabýval se vzhledem a designem pomocí tzv. **synth**. Vykreslování pomocí Java 2D nikoli pomocí bitmap. Integrováno do Java SE
- **swingLayout** – základna pro **`GroupLayout`**, který byl také integrován do Java SE 6
- **Scene graph** – knihovna poskytující nové funkcionality pro vzhled v Java 2D včetně Swing komponent. Stalo se základním stavebním kamenem platformy JavaFX
- **PDFRenderer** – knihovna poskytující zobrazování PDF
- **Swingset 3** – další generace demonstrativních programů, zobrazující nové funkcionality Swing
- **JXLayer** – universální dekoratér pro Swing komponenty. Integrován jako **`JLayer`** do Java SE 6

3.2 JavaFX

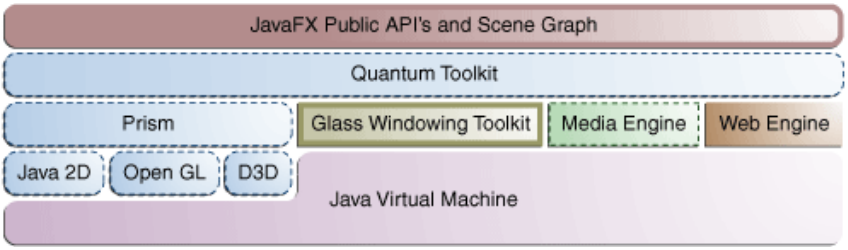
Je to softwarová platforma umožňující vytváření a poskytování tzv. Rich Internet Applications (RIA), tedy bohatých internetových aplikací, které můžou běžet na mnoha různých zařízeních. Je určená pro desktop, internetový prohlížeč i mobilní zařízení a její vývoj je pro všechna tato odvětví sjednocen. Poskytuje vývojářům bohatý balík grafických a mediálních API s vysoce výkonnými, hardwarově akcelerovanými grafickými a media enginy. JavaFX je psáno v Javě, takže Java vývojáři mohou zužitkovat své zkušenosti a nástroje pro vývoj JavaFX aplikací. JavaFX byla představena v roce 2007 a poskytovala JavaFX pro mobilní zařízení a *JavaFX Script* jazyk. V roce 2010 po odkoupení firmy Sun firmou Oracle byl vývoj *JavaFX Script* jazyku zastaven. Nejnovější verze JavaFX 2.0 má své vlastní API. Obsahuje nový grafický engine. Poskytuje nový deklarativní značkovací jazyk FXML založený na XML, který je možno využít pro definování uživatelského

rozhranní v JavaFX aplikacích. Obsahuje engine pro média, webovou komponentu, široký výběr ovládacích prvků uživatelského rozhranní atd.

Základní architektura JavaFX 2.0 platformy je zobrazena na obrázku *Obrázek 3.2*. Následující popis a obrázek architektury byl čerpán ze zdroje **[16]**.

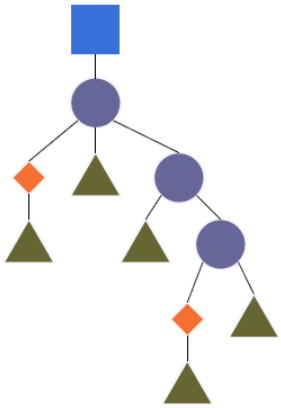
Pro tvorbu aplikace obsahuje JavaFX veřejné API. Obsahuje všechny potřebné třídy pro implementaci a kombinuje a doplňuje tak nejlepší vlastnosti Java platformy.

Obsahuje různé generické datové typy, rozšiřuje knihovnu Java kolekcí, umožňuje používání anotací, více vláknovou práci, událostní mechanismus a jiné.



Obrázek 3.2: Architektura JavaFX platformy

Hlavním stavebním kamenem pro konstrukci JavaFX aplikace je datová struktura **Scene Graph**. Ta představuje hierarchický strom složený z uzlů - *Node*. Ty reprezentují všechny vizuální prvky aplikačního uživatelského rozhranní. Každý uzel má své ID, třídu stylů a údaj o jeho ohraničení. Uzel má vždy jednoho rodiče a může mít několik potomků. `javafx.scene` API poskytuje vytvoření a specifikaci několika druhů obsahu. Kromě uzlů, jsou zde definovány stavy, kterých mohou uzly nabývat. Tím je myšleno různé geometrické transformace a jiné. Dále jsou nadefinovány efekty, které je možné jednotlivým prvkům přiřadit. Jsou to většinou efekty měnící vzhled uzlu, například stínování, změna barev, odraz apod. Efekty a transformace, které jsou přiřazeny uzlu, jsou při vykreslování scény aplikovány vždy na všechny potomky uzlu. Je tak možné provádět různé funkce souhrnně nad skupinou více prvků. Na obrázku *Obrázek 3.3* je zobrazena struktura Scene Graph.



Obrázek 3.3: Scene Graph

Tento obrázek byl získán ze zdroje **[17]**. Modrý čtverec v obrázku představuje kořen stromu scény. Fialové kruhy jsou rodičovské uzly, které v sobě obsahují více potomků. Oranžové čtverce představují stavový filtr, tedy nějakou transformaci, či efekt použitý na jeden samostatný uzel. Zelené trojůhelníky jsou listy stromu scény, tedy přímo komponenty Scene Graph, které neobsahují už žádné potomky. Mohou představovat různá geometriká primitiva, text, obrázky a jiné.

Uzlům lze také přiřadit událostní handlery, které zajišťují například události myši nad prvkem. Scene Graph obsahuje definici základních grafických primitiv, jako jsou čtverce, elipsy, text, různé kontrolní prvky, layoutovací kontejnery apod. Uzel může také obsahovat různé obrázky, nebo jiná média.

Vykreslování zajišťuje **grafický systém**, který je zakreslen na obrázku *Obrázek 3.2* pod vrstvou Scene Graph. Ten podporuje 2D i 3D. Poskytuje hardwarovou i softwarovou akceleraci vykreslování grafiky. *Quantum Toolkit* slouží tedy pro spojení *Prism* enginu a *Glass Windows Toolkit* a spravuje vlákna pro vykreslování a událostní správu.

Prism provádí vykreslování a může pracovat jak hardwarově, tak softwarově a to včetně 3D. Provádí rasterizaci scén. Pro svůj chod využívá různých zařízení. Pro operační systém Windows je to především DirectX. Pro ostatní je to převážně OpenGL. Je možné také využít Java2D, který sice nepodporuje hardwarovou akceleraci, avšak je možné ji využít, pokud jiné zařízení není k dispozici. Je totiž obsažena přímo v Java běhovém prostředí (JRE).

Glass Windowing Toolkit zodpovídá za poskytování nativních operačních služeb. To je například správa oken. Je to platformě závislá vrstva, která spojuje JavaFX platformu s nativním operačním systémem. Je také zodpovědný za událostní frontu. Na rozdíl od AWT však umožňuje správu i nativního událostního mechanismu. Toho se využívá pro plánování vláken. Glass Windowing Toolkit

běží na stejném vlákně jako JavaFX aplikace. Systém běží na několika vláknech. Vlákno JavaFX aplikace slouží především pro vyvíjení aplikace. Vlákno pro Prism slouží především pro renderování a je odděleno od správy událostí. Může obsahovat další pod-vlákna sloužící pro rasterizaci. Vlákno pro média běží na pozadí a synchronizuje obsah se scene graph skrze aplikační vlákno. *Pulse* vlákno zaručuje synchronizaci stavu elementů scene graph s Prism enginem. Je vždy vyvoláno, pokud běží nad scene graph nějaká animace a také je využíváno například při interakci s kontrolním prvkem. Díky tomu je možné asynchronně spravovat události v aplikaci.

JavaFX umožňuje práci s obrázky i s různými médii, jako je hudba či video skrze **`javafx.scene.media`** API. Dále obsahuje vestavěný webový prohlížeč s plnohodnotonou funkcionalitou. Je založený na `WebKit` a podporuje HTML5, CSS, JavaScript, DOM, SVG. Provádí renderování HTML obsahu z lokálního i vzdáleného URL. Podporuje ukládání historie i funkce *zpět* a *vpřed*. Umožňuje aplikaci efektů na webové komponenty, editaci HTML obsahu, vykonávání JavaScriptových příkazů a odchytávání událostí. Představuje tak komponenty **WebEngine**, který je obalen komponentou **WebView**, která zobrazuje obsah webových stránek.

JavaFX podporuje CSS nastavení stylů u uživatelského rozhranní bez změny zdrojových kódů aplikace. CSS mohou být aplikovány na jakýkoliv kořen JavaFX scene graph. Váhodou je, že mohou být přiřazeny za běhu aplikace. Založeny na W3C CSS verze 2.1 specifikaci.

Dále JavaFX poskytuje kontrolní prvky uživatelského rozhranní, které jsou dostupné skrze JavaFX API. Jsou postaveny na použití kořenů Scene graph. Díky CSS je možné měnit jejich vzhled. Umožňuje použití vestavěných layoutovacích modelů.

Distribuce JavaFX je možná ve třech módech.

- Samostatně fungující aplikace, která je nainstalována na lokální disk a běží jako spustitelný jar soubor, u kterého není potřeba být on-line.
- Uvnitř webového prohlížeče na webové stránce. Může pracovat s webovou stránkou skrze JavaScript.
- Pomocí WebStart, který stáhne aplikaci z webu a spustí jej na desktopu pomocí souboru JNLP.

JavaFX poskytuje vývoj plnohodnotných aplikací. Výhodnou je, že obsahuje mnoho nových technologií a především rozvinutý grafický engine, poskytující vytváření efektního grafického uživatelského rozhranní s rychlým hardwarovým vykreslováním. Další výhodou je, že je multiplatformní, takže můžou aplikace běžet na různých operačních systémech. Svojí funkcionalitou

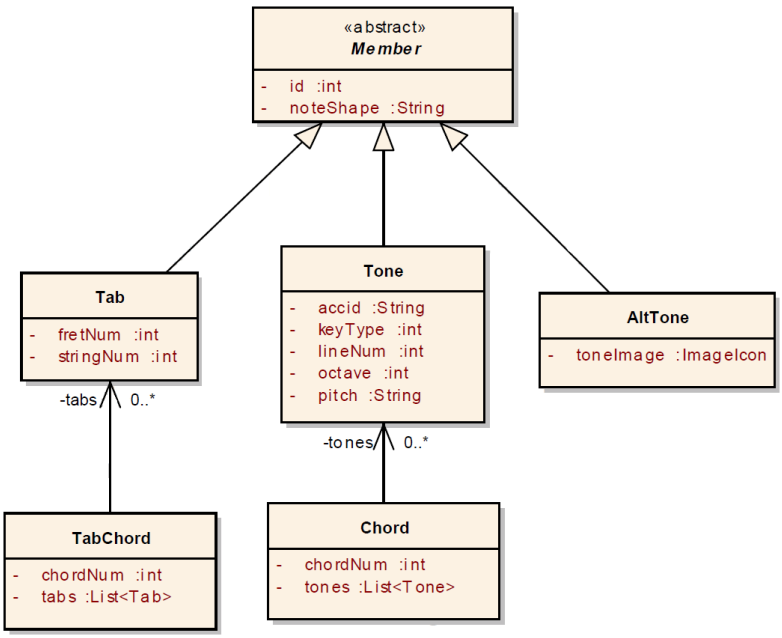
může konkurovat technologiím jako je Adobe FlashPlayer či Microsoft Silverlight. Nové informace, tutoriály a další užitečné věci lze nalézt na oficiálních stránkách JavaFX: **[18]**

4 Implementace aplikace

Tato kapitola se věnuje implementaci programu pro tvorbu a výuku hudby, který je součástí přílohy diplomové práce. Grafické uživatelské rozhranní je postavené především na Swing komponentách. Aplikace je tedy komponentově orientové. Využívá se funkcionality vestavěných kontrolních prvků, ale především jsou zde implementovány vlastní komponenty, vytvořené díky dědícímu mechanismu, který Java Swing podporuje. Třídy jsou pro přehlednost tématicky organizovány do jednotlivých balíků. Nejdříve budou popsány třídy sloužící pro syntetizaci hudby a různé hudební datové struktury, které budou potřebné pro celkovou funkci aplikace. V následujících kapitolách bude pak popsána struktura celé aplikace s jednotlivými částmi.

4.1 Hudební datové struktury

Pro potřeby aplikace bylo nutné jednotlivé hudební prvky popsané v teoretické části vyjádřit pomocí datových struktur. Provázanost jednotlivých prvků je vidět na UML třídního diagramu v obrázku *Obrázek 2.1*.



Obrázek 4.1: Třídní diagram hudebních prvků

Abstraktní třída **Member** představuje předpis pro jeden hudební prvek. Tím může být například tón, tab či jiná reprezentace. Atributem je zde `id`, tedy identifikační číslo prvku, které by mělo být pro celou hudební stopu unikátní. Podle tohoto id se pak prvek hledá. Unikátní id se získává pomocí volání statické metody `getNextId()` ve třídě `IdCounter`. Dále obsahuje parametr `noteShape`, představující tvar noty prvku a určuje tak délku tónu. Může nabývat hodnot: „W“ – nota celá (whole), „H“ – nota půlová (half), „Q“ – nota čtvrťová (quarter), „E“ – nota osminová (eighth), „S“ – nota šestnáctinová (sixteenth). Pomocí tohoto údaje a hodnoty tempa skladby se pak pro syntetizaci tónu spočte skutečná délka tónu v milisekundách.

Třída **Tone** dědí z Member. Představuje jeden hudební tón zobrazený pomocí noty. Parametr `pitch` může nabývat hodnot: {C, D, E, F, G, A, H}, a představuje tak jednotlivé tóny ze stupnice. Prvek **octave** pak naznačuje, ve které oktávě se vybraný tón nachází. Prvek **accid** (accidentials) slouží ke snížení či zvýšení tónu. Hodnoty jsou: „##” - zvýšení o dva půltóny, „#” - zvýšení o půltón, „b“ - snížení o půltón, „bb“ – zvýšení o dva půltóny, prázdná hodnota – žádné posunky. Poslední prvek **lineNum** značí číslo linky v notové osnově. Toto číslo se vypočítává z hodnot **pitch** a **octave**. Pokud je zadáno jen číslo linky, pak se naopak vypočítají zpětně hodnoty pitch a octave.

Třída **Chord** představuje jeden akord, který se skládá z jednotlivých tónů `Tone`, které při přehrávání zní jako souzvuk těchto tónů. Zní tedy ve stejný okamžik. Obsahuje prvek **chordNum**, tedy číslo akordu, což určuje umístění akordu v seznamu akordů v hudební stopě. Obsahuje seznam všech tónů `Tone`, které akord obsahuje. Jsou zde metody pro vkládání nových tónů a vymazávání obsažených tónů. Při vkládání nových tónů se ověřuje, jestli má tón stejný tvar noty jako ostatní tóny a také se zde provede kontrola unikátnosti tónu.

Třída **Tab** dědí z `Member`. Představuje jinou reprezentaci tónu a to pomocí tabulaturového kytarového zápisu. `Tab` je tedy prvek, který svoji výšku tónu určuje pomocí čísla struny kytary **stringNum** a čísla pražce **fretNum**, na kterém je struna stlačena. Číslo struny a číslo pražce však k určení tónu nestačí. Je nutné vědět, jaký tón má každá struna naladěna. K tomu slouží soustavy ladění definované pomocí třídy **Tuning**

Třída **TabChord** tak jako třída `Chord`, představuje jeden akord obsahující několik tabů tvořících souzvuk.

Třída **Tuning** reprezentuje ladění celého kytarového nástroje. Obsahuje název ladění **name** a seznam ladění jednotlivých strun **Tune**. `Tune` pak obsahuje parametr **stringNum**, což je číslo struny, která

má toto ladění. Dále obsahuje parametry **pitch** a **octave**, které dohromady určují tón, na kterou je daná struna naladěna. Ladění tedy určuje například pro 6ti strunnou kytaru 6 po sobě jdoucích tónů, představujících jednotlivé prázdné struny. Těchto ladění může být obrovské množství, avšak existuje standardní ladění(standard tuning), které používá většina hudebníků, či ladění, která jsou od tohoto standardního odvozena (např.:Drop-D tuning apod).

4.2 MIDI pomocné nástroje

Pro vytváření zvuků a tónů je využito prostředků z **javax.sound API**. Toto API je přizpůsobeno pro práci Java aplikací s hudbou. Poskytuje nástroje, pomocí kterých je možné vytvářet hudbu díky systému MIDI. Vytváření hudby se zde neprovádí přehráváním hudební stopy, ale využívá se zvukových bank a díky syntetizátoru je možné na jeden z jeho deseti kanálů, který má načtený příslušný instrument ze zvukové banky, posílat MIDI zprávy. Podle zaslaných kanálových zpráv vytváří MIDI zařízení příslušný zvuk ze zvukové banky.

Třída **MidiSynthesizer** obaluje práci se třídou **Synthesizer.** Představuje tak objekt, který poskytuje midi kanály, pomocí kterých je možné syntetizovat zvuk na výstupním zařízení. K metodám této třídy se přistupuje výhradně staticky, takže je možné využívat midi zdroje globálně v celé aplikaci. Pro umožnění práce je nejdříve nutné jej otevřít. To se provede metodou **open()**. Zde se nejdříve získá nový **Synthesizer** pomocí **MidiSystem.getSynthesizer()**. Následně se syntetizátor otevře. Získá se zvuková banka **Soundbank**, která obsahuje sadu zvuků, které je možné pomocí syntetizátoru vytvářet. Z banky jsou získány všechny hudební nástroje **Instrument** do pole. Ze syntetizátoru se pak získá pole midi kanálů **MidiChannel**, z nich se pak sestaví pole kanálových dat **ChannelData**. Metodou **close()** je možné syntetizátor uzavřít. Pomocí metody **getChannel** se získá objekt kanálových dat `ChannelData`.

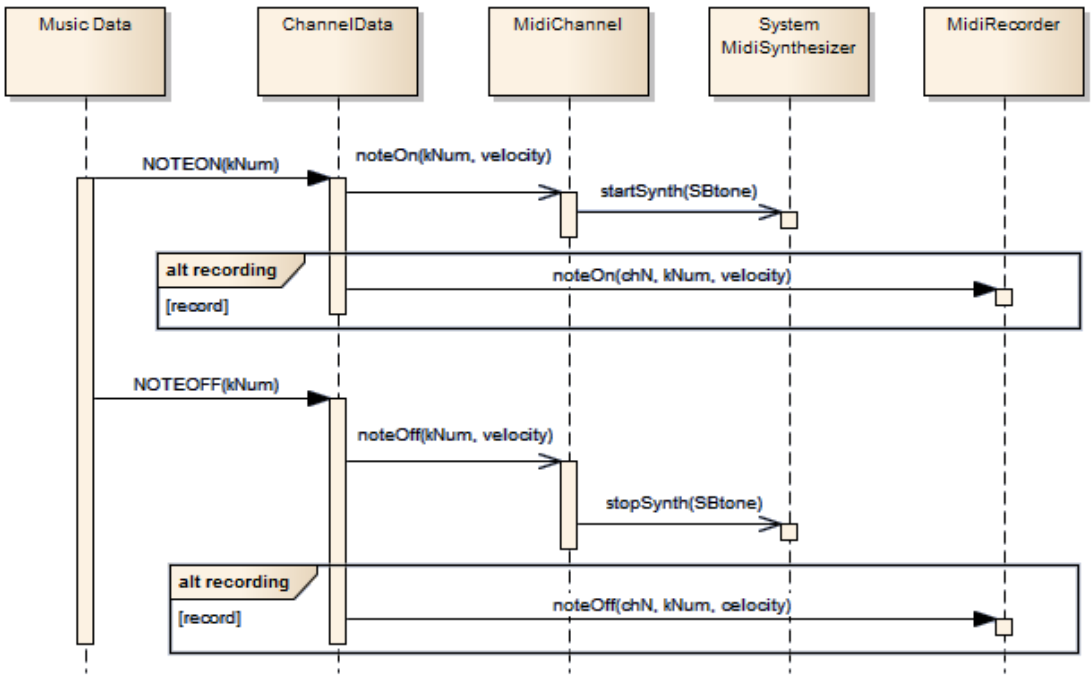
Třída **MidiRecorder** Slouží pro záznam kanálových dat do objektu **Sequence**. Pro spuštění záznamu slouží statická metoda **record()**. V ní se vytvoří nová sekvence, do které se budou zaznamenávat midi zprávy. Poté se získá ze sekvence nová stopa **Track**, která bude představovat právě jednu stopu sekvence. Pak se nastaví počáteční čas záznamu **startTime**. Metoda **createShortEvent** slouží pro vytvoření nové události, která se zaznamená do stopy Track. Události je nutné předat číslo kanálu, pro který platí tato událost, dále typ kanálové zprávy, která se má zaznamenat, případně číslo zaznamenávaného tónu a jeho síla. Této události se využívá buď při

započetí nového záznamu pro inicializaci kanálů ve stopě, ale hlavně záznam událostí nad kanálem, tedy metody **noteOn** a **noteOff**. V nich se uplatní číslo kanálu a jeho síla. Je nutné vždy při záznamu této zprávy zjistit stávající čas od počátku záznamu a podle toho spočíst počet úderů, které uběhly. Vytvořená událost **MidiEvent** se nakonec přiloží do stopy **Track**. Metoda **exportToMidi** získá ze všech hudebních stop, které nejsou utlumené, sekvenci záznamu celé skladby. Slouží tedy pro export hudebních dat do hudebního souboru.

Dále obsahuje pohled **MidiRecorderView**, představující grafické uživatelské rozhranní pro práci s **MidiRecorder**. Obsahuje tlačítko **recordBtn** pro započetí záznamu a ukončení záznamu. Při ukončení záznamu se automaticky nová sekvence uloží do seznamu sekvencí. Tento seznam sekvencí je zobrazen pomocí komponenty `JList`. Díky objektu Sequnecer je možné pak vybranou sekvenci přehrát stlačením tlačítka `playBtn`. Stlačením tlačítka `saveBtn` se dá vybraná sekvence uložit do souboru ve formátu **.mid**.

Třída **ChannelData** představuje třídu, která obaluje práci s MIDI kanálem **MidiChannel**. Hlavní funkcí je syntetizování zvuků skrze MIDI kanál na systémovém MIDI syntetizátoru . To se provádí pomocí metod **NOTEON** a **NOTEOFF**. Metoda `NOTEON` slouží pro započetí znění vybraného tónu z kanálu, definovaného parametrem **kNum**. Pro ukončení znění daného tónu je pak potřebné na tomto kanále zavolat metodu `NOTEOFF` opět s číslem tónu z kanálu. Na obrázku sekvenčního diagramu *Obrázek 4.2* je vyznačeno volání metod a zasílání kanálových zpráv skrze Midi kanál na systémový syntetizátor. Je zde i zahrnut záznam Midi sekvence pomocí `MidiRecorder`.

Kanálovým datům je možné nastavovat několik parametrů jako je **velocity** což je síla úderu tónu, dále **reverb**, což představuje určitou míru ozvěny tónu. Dále je možné kanál nastavit jako utlumený pomocí **setMute**, či lze nastavit kanál jako sólový **setSolo**, tzn. že ty kanály, které jsou sólové budou hrát, ostatní kanály budou utlumené, aniž by měly nastaven příznak *mute*. Protože kanálová data také obsahují seznam instrumentů, které se vyskytují ve zvukové bance, je možné díky nim změnit program kanálu na vybraný instrument metodou **programChange**. Dále je možné nastavovat parametry přímo kanálu jak jsou **pressure** – nastavení tlaku na hraný tón, **bend** – představující vyhnutí tónu o určitou výšku nahoru či dolu. Pokud je zvoleno nahrávání všech MIDI zvuků v rámci všech kanálů, pak se v metodách `NOTEON`, `NOTEOFF` a programChange provede přenesení těchto kanálových zpráv i na objekt sloužící pro zaznamenávání těchto kanálových zpráv **MidiRecorder**. ChannelData dále obsahuje svůj pohled **ChannelDataView**, což je grafická komponenta představující uživatelské rozhranní pro změnu parametrů tohoto kanálu. Obsahuje



Obrázek 4.2: Sekvenčí diagram zobrazující posílání kanálových zpráv mezi MIDI zařízeními

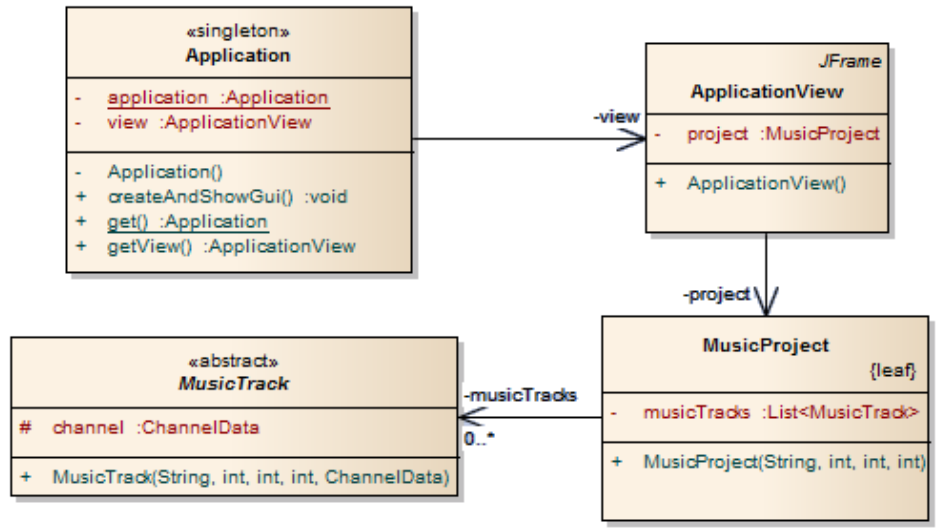
ovládací prvky **`JSlider`**, pomocí kterých je možné nastavovat parametry `velocity`, `pressure`, `reverb` a `bend`. Dále obsahuje tabulku `JTable`, ve které jsou zobrazeny všechny dostupné instrumenty. Výběrem jednoho z instrumentů se kanálu tento instrument změní a tím dochází k jiné syntetizaci zvuku.

4.3 Základní struktura celé aplikace

Na následujícím obrázku třídního diagramu *Obrázek 4.3* je zobrazena základní struktura aplikace. Podrobnější struktura jednotlivých částí aplikace bude zobrazena dále.

Aplikace je spouštěna pomocí hlavní třídy **`MusicOnLine`**, ve které se nachází hlavní spouštěcí metoda, pomocí které se aplikace vytvoří a spustí.

Třída **`Application`** tvoří jádro celé aplikace. Je implementována pomocí návrhového vzoru ***Singleton***, tedy jako tzv. „jedináček“. To znamená, že třída bude mít pouze jedinou instanci a je možné k ní přistupovat globálně pomocí statického přístupu metody **`get`**. Grafické uživatelské rozhranní celé



Obrázek 4.3: Základní struktura aplikace

aplikace a většina funkcionality je soustředěna do třídy **`ApplicationView`**. To se vytvoří a zobrazí pomocí metody **`createAndShowGui`**.

Třída **`ApplicationView`** dědí z grafické komponenty **`JFrame`**. Tvoří tedy hlavní okenní rám aplikace. Při inicializaci je vytvořeno menu aplikace díky komponentě **`JMenuBar`**. Menu obsahuje všechny potřebné položky pro práci s aplikací. Menu je pak přiřazeno přímo rámu. Výplň rámu má nastavené rozvržení komponent pomocí instance třídy **`BorderLayout`**. Celou výplň pak tvoří komponenta **`JDesktopPane`** zarovnaná na střed. `JDesktopPane` umožňuje vytvářet vnitřní rámy **`JInternalFrame`**, se kterými lze pracovat pouze uvnitř desktopu. Vnitřní rámy je možné zavírat, maximalizovat, ikonifikovat, měnit velikost atd. tak jako normální okna aplikací. Uvnitř desktopu se tedy zobrazují vnitřní rámy většiny grafických komponent, se kterými je možné v aplikaci pracovat. ApplicationView obsahuje instanci **`MusicProject`**, který představuje jeden hudební projekt, pomocí kterého se tvoří jednotlivé hudební stopy. Při vytvoření nového projektu `MusicProject` se jeho grafická komponenta zobrazí v novém vnitřním rámu. S tím se vytvoří i nástrojový panel **`JToolBar`**. Ten obsahuje všechny důležité ovládací prvky, díky kterým je možné pracovat s projektem. Jsou to tlačítka pro spouštění či zastavování přehrávání skladeb, ukládání stávajícího projektu a jiné. `JToolBar` má dobrou vlastnost a to tu, že může být přidán do hlavního rámu aplikace, který má nastavené rozvržení pomocí `BorderLayout`. Díky tomu, že je JDesktopPane zarovnán na střed, může být nástrojový panel umístěn na místo okolo `JDesktopPane`. Panel je pak

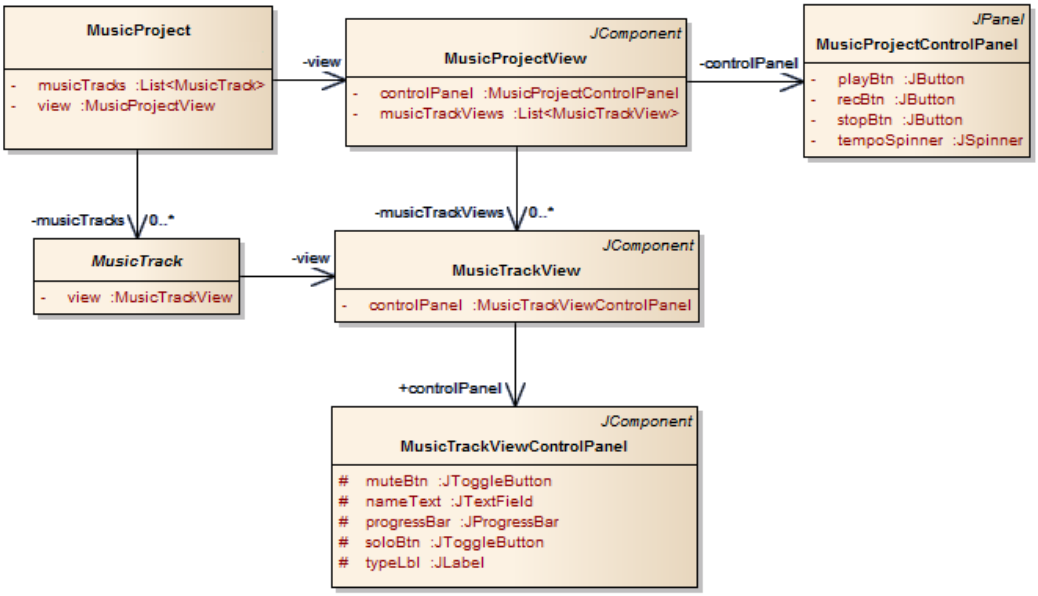
možné přesouvat vždy k jednomu z okrajů tedy hornímu, dolnímu, levému či pravému. Také je možné panel vytáhnout mimo rám, pak se tedy vytvoří samotné okno, ve kterém je zobrazen pouze panel s jeho ovládacími prvky. Nakonec je možné jej vrátit zpět do hlavního rámu přetažením panelu, či zavřením jeho okna.

Třída **`MusicProject`** představuje hudební projekt, ve kterém je zahrnuto vytváření a práce s hudebními stopami a další funkce. V rámci celé aplikace může být maximálně jeden otevřený projekt. Projekt má své id, jméno, obsahuje nastavení hodnot tempa skladeb a typu úderů. Dále obsahuje grafické rozhranní reprezentované komponentou **`MusicProjectView`**. Nejdůležitější součástí je však seznam hudebních stop **`MusicTrack`**.

Třída **`MusicProjectView`** slouží k zobrazení přehledu jednotlivých hudebních stop. V této třídě dochází k nejdůležitějším operacím při ovládání uživatelského rozhranní v rámci hudebních stop. Obsahuje seznam ovládacích náhledů na hudební stopy **`MusicTrackView`**, ty se pak zobrazují postupně shora dolů v rolovacím panelu **`JScrollPane`**. Tyto pohledy je možné myší seřazovat a ovládat jejich jednotlivé kontrolní prvky a zobrazovat tak například pohledy přímo na stopy pomocí notových nebo jiných osnov.

Abstraktní třída **`MusicTrack`** představuje jednu hudební stopu se kterou je možné pracovat. Má své jméno, identifikační číslo, hodnotu tempa stopy, označení typu stopy a přidělený kanál **`ChannelData`**, pomocí kterého se hudební stopa syntetizuje. Dále obsahuje svůj pohled **`MusicTrackView`**. Tato hudební stopa však ještě neobsahuje žádná hudební data, která by se dala syntetizovat. Ty je nutné naimplementovat až v nějaké konkrétní zděděné podtřídě. Třída má předpřipravené abstraktní metody **`play()`**, **`stop()`**, **`prepare()`**, **`isSolo()`**, **`setSolo()`**, **`isMute()`**, **`setMute()`**, které musí konkrétní podtřídy implementovat, protože slouží k jednotnému ovládání stopy jako je přehrávání či nastavení různých parametrů.

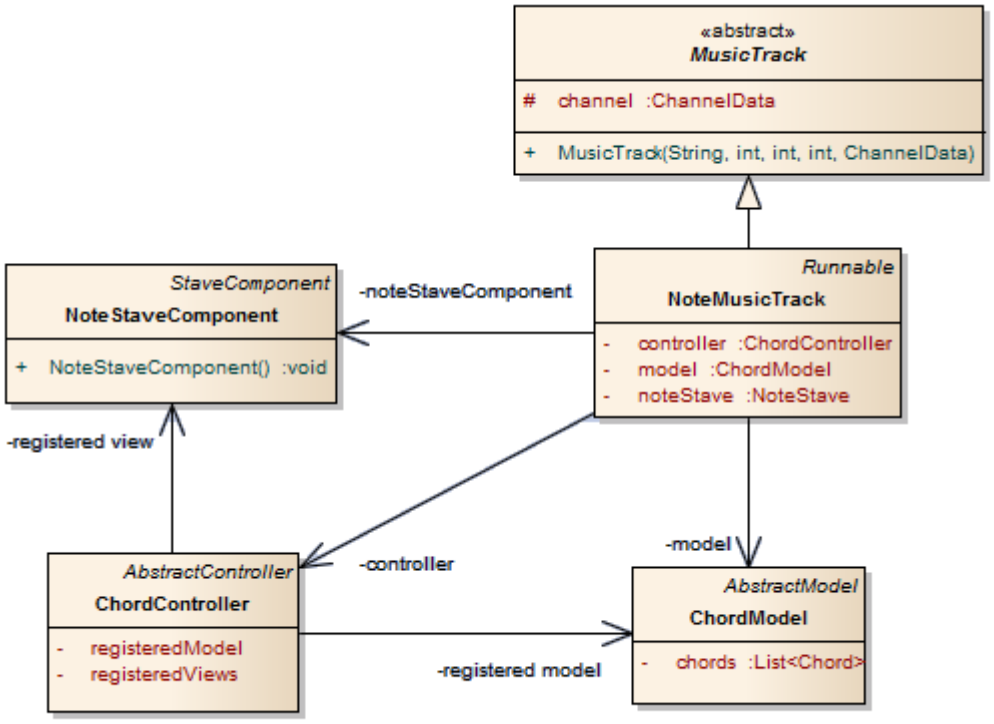
Třída **`MusicTrackView`** představuje grafický předběžný a ovládací náhled na hudební stopu, ke které náleží. Obsahuje kontrolní panel s tlačítky pro ovládání stopy, vypínání stopy, tlačítka pro zobrazení náhledů a editorů osnov, zadání jména stopy apod. Dále obsahuje zmenšený náhled na hlavní osnovu, u notové hudební stopy **`NoteMusicTrack`** je realizován náhledem notovou osnovu `NoteStavePlayer`. U bicích hudebních stop žádný náhled není. Na obrázku *Obrázek 4.4* je zobrazen třídní diagram rozložení hudebního projektu a hudebních stop a umístění jejich pohledů a kontrolních panelů. V následujících kapitolách budou popsány konkrétní typy hudební stopy, které jsou v aplikaci implementovány.



Obrázek 4.4: Třídní diagram hlavního grafického uživatelského rozhranní

4.4 Notová hudební stopa

Notovou hudební stopu představuje třída **`NoteMusicTrack`**, která dědí ze třídy **`MusicTrack`**. Představuje již konkrétní hudební stopu obsahující hudební data. Hudebními daty tu jsou tóny **`Tone`**, které jsou poskládány do akordů **`Chord`** a tyto akordy pak jsou seřazeny do seznamu. Na tuto stopu akordů by mělo být možno pohlížet z několika různých pohledů, pomocí kterých by bylo možné stopu měnit. Příkladem je zde klasická notová osnova, ale pak i třeba tabulaturová osnova pro zápis not pomocí kytarových značek a mělo by být možné přidat případně i jiný alternativní náhled, určený pro jiný zápis specifický pro nějaký nástroj. Tento problém několika pohledů a jejich synchronizace nejlépe řeší návrhový vzor **Model-View-Controller**. Implementace MVC pro tento konkrétní problém je popsána v následujících podkapitolách. Na obrázku třídním diagramu *Obrázek 4.5* je zobrazena základní struktura notové hudební stopy.



Obrázek 4.5: Třídní diagram zobrazující strukturu notové hudební stopy

4.4.1 Model – View – Controller architektura

Architektura je dána třemi abstraktními prvky `AbstractModel`, `AbstractView` a `AbstractController`. Ty definují základní strukturu. Pro využití funkcionality MVC pak budou tyto abstraktní prvky děděny a implementovány pro konkrétní třídy. Bylo využito typu MVC architektury, kdy je **Controller** postaven mezi **Model** a **View**, které spolu komunikují výhradně skrze Controller.

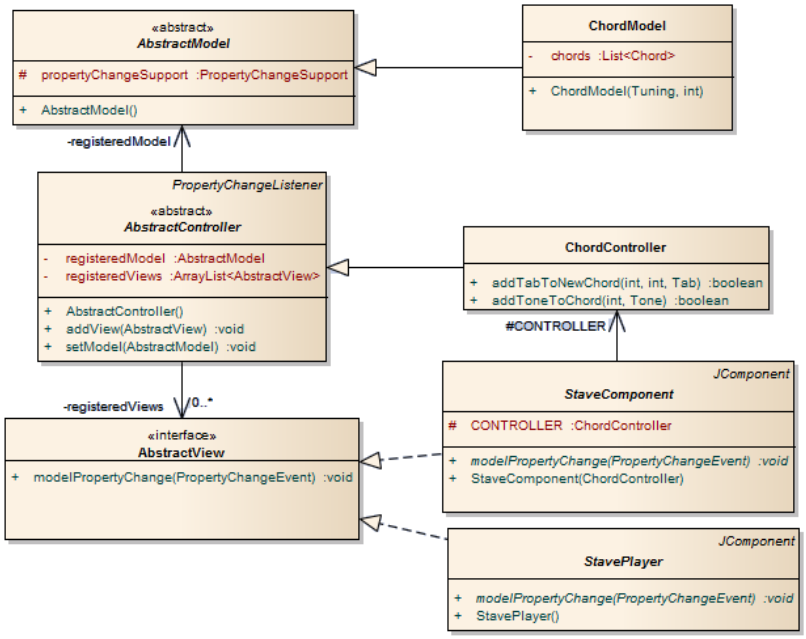
Abstraktní třída **`AbstractModel`**, představuje abstraktní model, jehož implementace bude obsahovat hudební data. Obsahuje instanci **`PropertyChangeSupport`**, která poskytuje podporu pro změnu vlastností modelu. Může se jí přiřadit **`PropertyChangeListener`**, který naslouchá změnám modelu. Při změně dat v modelu se pak vyvolá událost, která změní všechny pohledy pomocí metody **`firePropertyChange`**.

Rozhranní **`AbstractView`** představuje definici pohledu, který je připojen k modelu. Obsahuje definici pouze jedné metody **`modelPropertyChange`**, která je volána vyvoláním události změny dat v modelu. Pro změnu pohledu je pak nutné v konkrétním pohledu provést vlastní implementaci.

Abstraktní třída **`AbstractController`** představuje kontrolora, který obstarává změnu pohledů podle změny modelu a naopak. Implementuje rozhranní **`PropertyChangeListener`**, což znamená, že v této třídě je možné naslouchat události vyvolané změnami na modelu. Kontrolor obsahuje seznam připojených pohledů, které implementují rozhranní `AbstractView` a dále registrovaný model, který musí dědit z `AbstractModel`. Pomocí metody **`setModel`** se nastaví kontroloru model, kterému se přiřadí posluchač pro změnu dat v modelu. Metoda **`addView`** přiřadí nový pohled. Události vyvolané v modelu se v kontroloru poslouchají implementovanou metodou **`propertyChange`** z **`PropertyChangeListener`** posluchače. Zde se projdou všechny připojené pohledy a pro každý z nich se vyvolá metoda **`modelPropertyChange`**, pomocí které jsou předány informace o změněné vlastnosti modelu. Pro změnu modelu z pohledu se volá metoda **`setModelProperty`**, která na modelu vyvolá příslušnou metodu, pomocí které se provede změna vybrané vlastnosti.

4.4.2 Konkrétní implementace Model – View – Controller

Vpředchozí kapitole byla popsána struktura MVC, které bude využito pro konkrétní implementaci. Struktura implementované MVC architektury je zobrazena na obrázku třídního diagramu *Obrázek 4.6*.



Obrázek 4.6: Třídní diagram implementovaného Model-View-Controller

Třída **ChordModel** představuje konkrétní model obsahující data hudební stopy v podobě seznamu akordů **Chord**. Dědí z `AbstractModel`. Výchozím modelem je tedy notový zápis. Tento model obsahuje všechny potřebné metody určené pro změnu dat v modelu. Při úspěšné změně modelu je pak nutné tuto změnu provést i na ostatních připojených pohledech. To se provede vyvoláním události změny vlastnosti modelu **firePropertyChange** z nadtřídy **AbstractModel**.

Třída **ChordController** představuje konkrétní kontrolor. Jeho instance se nachází v každém pohledu, který má být připojen k modelu. Pro změnu dat v modelu z pohledu je nutné zavolat příslušnou metodu, která pomocí metody **setModelProperty** provede změnu vlastnosti v připojeném modelu.

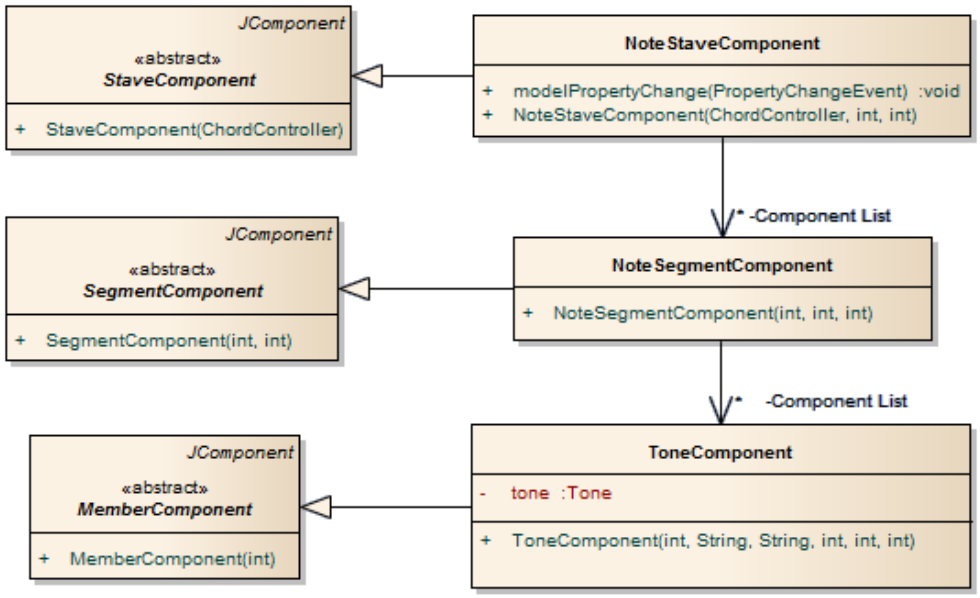
Třída **StaveComponent** představuje nadtřídu pro komponenty, které se chovají jako pohledy, pomocí kterých je možné měnit vlastnosti modelu, tedy editory. Implementuje tak rozhranní **AbstractView**. Obsahuje instanci **ChordController**, díky kterému se provádí změny modelu zavoláním příslušné metody. Struktura komponenty bude popsána v následující kapitole.

Třída **StavePlayer** představuje nadtřídu pro komponenty, které se chovají také jako pohledy. Tyto však zobrazují pouze obsah modelu a není možné provádět změnu dat. Slouží pro zobrazení aktuálně hraného tónu/ akordu při přehrávání notové hudební stopy.

4.4.3 Notové a tabulaturové osnovy

Třída **StaveComponent** představuje abstraktní osnovu, jejíž konkrétní implementace budou sloužit pro zápis například not či tabů. Dědí z **JComponent**, takže je možné na ni pohlížet jako grafickou komponentu pro vykreslování grafického obsahu. Celý obsah je tvořen po sobě jdoucími komponentami **SegmentComponent**. Ty představují časové úseky, do kterých se zapisují prvky (noty, taby). Prvky, které se nacházejí v jednom úseku tvoří souzvuk tónů, protože při přehrávání hudební stopy zní současně. Je to tedy akord. Úseky jsou identifikovány pomocí čísla **segmentNum**, díky kterému je možné seřadit jednotlivé úseky postupně zleva doprava pomocí třídy **SegmentLayout** z rozhranní `LayoutManager`, sloužící pro layoutování komponent uvnitř kontejneru. Dále obsahuje číslo **chordNum**, které určuje číslo akordu v modelu. Pokud segment neobsahuje žádný tón, pak je tato hodnota nastavena na - 1. Prvek, který může být přidán do úseku SegmentComponent je **MemberComponent**.

Na obrázku *Obrázek 4.7* je zobrazen třídní diagram, který představuje strukturu abstraktní osnovy s jejími abstraktními prvky a dále příslušné třídy, které implementují konkrétní notovou osnovu. Tabulaturová osnova má svoji implementaci obdobnou.



Obrázek 4.7: Třídní diagram zobrazující strukturu notové osnovy

Třída **`NoteStaveComponent`** je implementace konkrétní notové osnovy, která dědí ze **`StaveComponent`**. Její obsah je tvořen úseky **`NoteSegmentComponent`** dědící ze SegmentComponent. Do úseků je pak možné vkládat nové prvky, v tomto případě komponenty **`ToneComponent`**, představující noty. ToneComponent Dědí z **`MemberComponent`**. Obsahuje záznam o tónu **`Tone`**, který tato komponenta má v notové osnově představovat. Její obsah je vykreslován pomocí komponenty **`NoteShapeComponent`** představující tvar noty a komponenty **`AccidShape`**, která vykresluje zvýšení či snížení noty. Podle typu notové osnovy, který se může měnit podle nastavení notového klíče, se seřazují noty v jednotlivých úsecích pomocí **`NoteSegmentLayout`**. Ten seřadí všechny noty pomocí jejich parametrů výšky tak, aby byly na, nebo mezi linkami **`LineComponent`**, podle pravidel zápisu not do notové osnovy.

Třída **`TabStaveComponent`** je implementace konkrétní tabulaturové osnovy. Také dědí ze **`StaveComponent`**. Její obsah je tvořen úseky **`TabSegmentComponent`**. Do úseků se vkládají

nové prvky **TabComponent** dědící z **MemberComponent**. Obsahuje záznam o konkrétním tabu **Tab**, který má komponenta představovat. Obsah komponenty je vykreslován jako čtverec s číslem pražce tabu uprostřed. Tabová osnova obsahuje tolik linek **LineComponent**, kolik strun má kytarový nástroj, ke kterému se daný zápis vztahuje. Typicky je to pro kytaru šest linek a pro basovou kytaru čtyři linky. Podle čísla struny v tabu **stringNum** se tabové komponenty seřazují v úsecích pomocí **TabSegmentLayout**. Na každé lince v daném úseku se může nacházet maximálně jeden tab.V notové i tabulaturové osnově je možné vkládat nové prvky kliknutím do některého ze segmentů. Podle pozice linky, na kterou bylo kliknuto se vloží příslušný prvek s určitými parametry, danými počátečním nastavením, které je možné měnit a podle linky, na které se prvek nachází. U tabů je rozhodující číslo linky, protože představuje číslo struny, které spolu s číslem pražce určuje výšku tónu tabu. U noty poloha na lince také ovlivňuje výšku tónu. Z této pozice je vypočtena hodnota **pitch** a **octave** a spolu s nastavením posunků **accid**, je určena výška tónu noty. Prvky je možné přetahovat myší na jiné linky i do jiných úseků. Celé úseky je také možné seřazovat i s prvky uvnitř. Vždy po přidání prvků do pohledů, či změně jednoho z pohledů, se provede změna modelu s daty a automaticky jsou změněny i ostatní připojené pohledy. S tím i souvisí převod mezi jednotlivými prvky, který není triviální. Tyto převody se provádějí v modelu díky statickým metodám ze třídy MusicBox, jejíž funkcionalita bude popsána později.

4.4.4 Notové a tabulaturové přehrávače

Předchozí osnovové komponenty představovaly editorové zápisy. Pro zobrazení a zvýrazění právě přehrávaných tónů/ akordů slouží následující přehrávače.

Třída **StavePlayer** představuje abstraktní definici struktury přehrávače osnov. Implementuje rozhranní **AbstractView**, takže se jedná opět o pohled na modelová data. Tak jako **StaveComponent** obsahuje seznam úseků z nadtřídy **SegmentComponent**. Jsou zde však pouze segmenty, které obsahují alespoň jeden prvek. Úseky jsou seřazeny vedle sebe podle čísla **chordNum**, namísto **segmentNum**. Pozice celého zápisu je určena aktuální přehrávací pozicí **actPosX**. Podle toho jsou úseky umístěny na ose X. Aktuálně hraný akord se nachází v šedém pruhu vyznačeném uprostřed komponenty. Přehrávací pozice se mění pomocí metody **setActChord**, která nastaví podle čísla akordu chordNum aktuálně přehrávaný akord. Toho využívá metoda **moveToNext**, která posune osnovu o jednu pozici akordu dále. Metoda **setToBegin** umístí osnovu na první akord. Přechod mezi jednotlivými akordy je možné animovat. To se provede v metodě **animate**. Pokud je nastaven příznak pro animaci, pak se pomocí časovače **Timer** postupně v jednotlivých krocích přičítá

hodnota aktuální pozice `actPosX` a komponenta se vykresluje. To má za výsledek plynulý přechod z jednoho akordu na druhý, oproti skokovému, v případě nenastavení příznaku pro animaci.

Třída **`NoteStavePlayer`** a **`TabStavePlayer`** představují konkrétní implementované přehrávače, které jsou určené pro notovou osnovu a tabuluturovou osnovu. Jejich obsah tvoří tak jako u editovatelných pohledů **`NoteSegmentComponent`** komponenty pro notovou osnovu a **`TabSegmentComponent`** komponenty pro tabulaturovou osnovu. Mají implementovanou metodu z nadtřídy **`modelPropertyChange`** sloužící pro změnu pohledu vzhledem k změněným datům v modelu.

4.4.5 Spojení editorů osnov a přehrávačů osnov

Protože editory a přehrávače pro osnovy jsou obdobné, i když mají každý jiný účel, jsou tyto dva pohledy spojeny do jedné komponenty. Pro notové pohledy je to **`NoteStave`**, která dědí z **`JTabbedPane`**. To znamená, že mezi osnovami, které jsou umístěny do tohoto kontejneru je možné přepínat pomocí záložek. Nejsou sice viditelné v jeden okamžik, což však ani není nutné vzhledem k účelu každé z komponent a není nutné je zobrazovat ve stejnou dobu. Pro editor notové osnovy NoteStaveComponent je zde navíc umístěn kontrolní panel **`controlPanel`** sloužící pro nastavení aktuálně vkládané noty do osnovy a panel **`propertiesPanel`** pro změnu vlastností vybrané noty, či její mazání.

Pro tabulaturové pohledy je to **`TabStave`**, která má stejnou funkci jako **`NoteStave`**, tedy pomocí dědění z **`JTabbedPane`** poskytuje oddělení editace od přehrávání tabulatury. Editovatelná komponenta také obsahuje **`controlPanel`** a **`propertiesPanel`** pro ovládání.

4.4.6 Alternativní náhledy na notovou osnovu

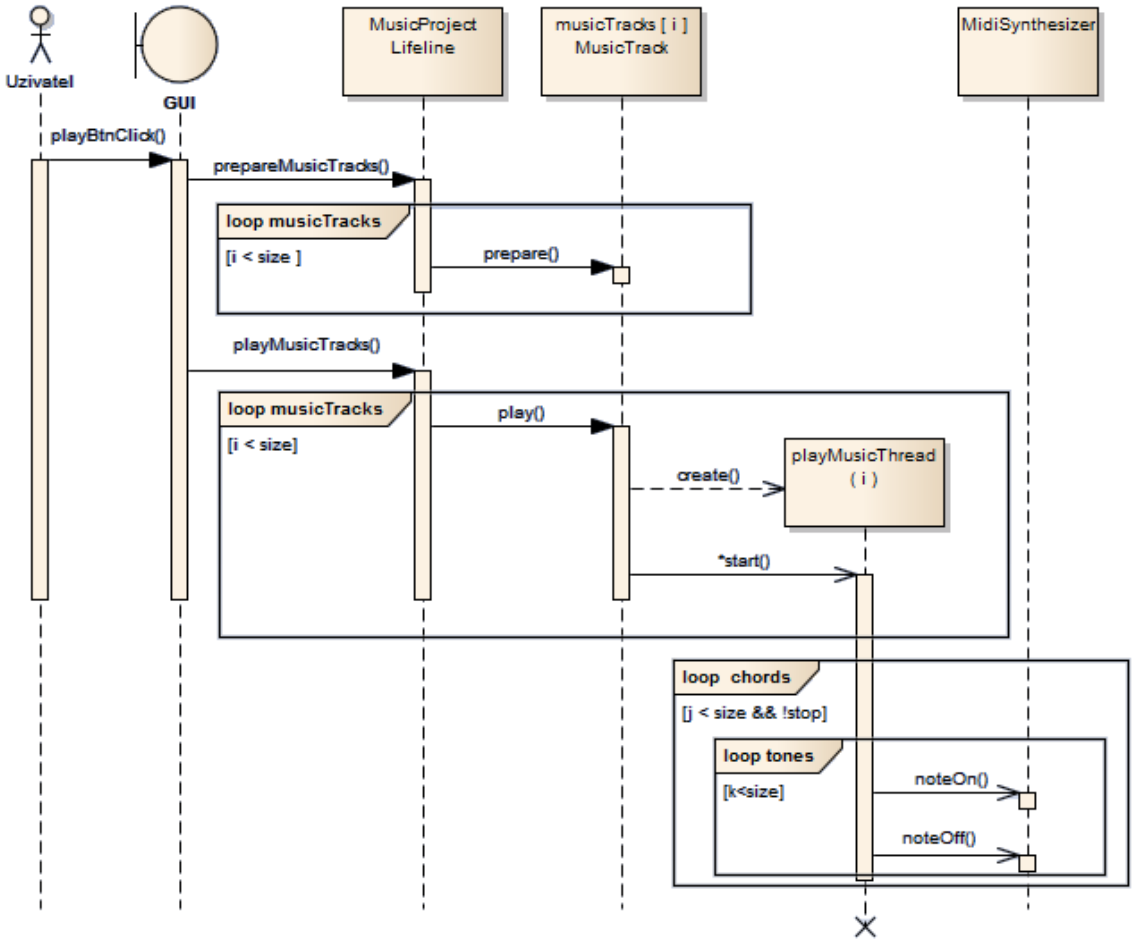
Jedním z alternativních zápisů na notovou osnovu byl již uveden zápis pomocí tabulatury určené pro kytarový nástroj. Avšak notovou osnovu by mělo být možné zobrazit i pomocí dalších náhledů, které mohou příslušet i jiným hudebním nástrojům, pro které je jejich zápis uzpůsoben. Příkladem může být zakreslení určitého tónu pomocí obrázku flétny s vyznačenými dírkami, které mají být zacpány prsty při foukání. Nejlepším řešením takovéto alternativní notové osnovy, která by mohla zobrazit jakýkoliv zápis, je použití sady obrázků, kde každý obrázek by představoval jeden konkrétní tón. Protože rozsah tónů použitelných pro většinu hudebních nástrojů je od druhé do šesté oktávy, kde každá oktáva obsahuje dvanáct půltónů, pak je potřeba mít maximálně 60 obrázků pro zápis jednoho nástroje. Ve většině případů je tento rozsah však i mnohem menší.

Pro implementaci alternativní notové osnovy zde slouží komponenta **AltStaveComponent**. Ta implementuje rozhranní **AbstractView**, aby bylo možné s komponentou také pracovat jako s pohledem. Tím je implementována metoda **modelPropertyChange**, která je volána vždy při změně modelu dat. Pro zobrazení samotného alternativního tónu zde slouží **AltToneComponent**. Ta v sobě zobrazuje přiřazenou obrázkovou ikonu alternativního tónu. Pro vyřešení souzvuků alternativních tónů, je zde **AltSegmentComponent**, která v sobě řadí jednotlivé AltToneComponent, od shora dolů. Tyto segmenty souzvuků jsou pak přiřazeny do **AltStaveComponent**, ve které jsou seřazeny podle čísla akordu. Pro získání obrázkové ikony do alternativního tónu zde slouží **AltToneFactory**. Ta obsahuje hashovací mapu **HashMap**, která v sobě nese sadu klíčů a k nim přiřazených hodnot, v tomto případě je klíčem vždy unikátní název tónu. Například tón C se zvýšením o půltón a nacházející se ve třetí oktávě má klíč „C#3“. Hodnotou, na kterou klíč odkazuje je textový řetězec URL cesty k obrázku příslušného alternativního tónu. Pomocí metody **getAltToneImageIcon** je možné získat na základě předaných parametrů odkaz na URL obrázku a vrátit novou obrázkovou ikonu. Pro sestavení hashovací mapy je však nutné znát tyto URL odkazy. Ty se získají pomocí definičního souboru, který má uloženy odkazy na všechny obrázky tónů. Tento soubor je strukturován pomocí XML jazyka. Pro načtení nové mapy je nejdříve nutné vybrat příslušný definiční soubor, ze kterého se díky pomocné třídě **MusicXmlManager** metodou **getAltTones** vytvoří nová hashovací mapa. V definičním souboru jsou pak uložené pouze relativní adresy všech tónů a absolutní cesta se pak sestaví z pozice definičního souboru. Při načtení nových alternativních tónů se provede obnovení osnovy a pokud model již obsahuje vytvořené tóny, pak se tyto tóny zobrazí i v alternativní osnově. Jakákoliv změna modelu se tak jako u ostatní pohledů projeví i zde. Obrázek

4.4.7 Celková funkce notové hudební stopy

V předchozích kapitolách byly definovány všechny pohledy, které je možné vytvářet pro notovu hudební stopu. Tato stopa NoteMusicTrack udržuje svá hudební data v modelu akordů **ChordModel**. Dále obsahuje všechny pohledy, které je možné tomuto modelu přiřadit. Patří mezi ně NoteStaveComponent, NoteStavePlayer, TabStaveComponent, TabStavePlayer, AltStaveComponent. Pro řízení jednotlivých pohledů a komunikaci s modelem zde slouží **ChordController**, Tomuto kontroloru jsou přiřazeny všechny pohledy a samotný model. Proces přehrávání hudebních stop je zobrazen na sekvenčním diagramu na obrázku *Obrázek 4.8*.

Přehrávání všech hudebních stop se provede stisknutím tlačítka **`playBtn`** v nástrojovém panelu. Tím se ve třídě **`MusicProject`** zavolá nejdříve metoda **`prepareMusicTracks`**, která připraví všechny hudební stopy, které se v projektu nacházejí. To se provede pro každou hudební stopu v implementované metodě **`prepare()`**, ve které se připraví model a jeho data, která se v něm nacházejí a dále všechny osnovy, které zobrazují aktuálně přehrávané akordy se nastaví na začátek. Následně je zavolána metoda **`playMusicTracks`**, která spustí přehrávání jednotlivých stop.



Obrázek 4.8: Sekvenční diagram přehrávání hudebních stop

Protože NoteMusicTrack implementuje rozhranní Runnable, je možné se stopou pracovat jako s vláknem, takže je možné pustit několik hudebních stop, představujících jednotlivé hudební nástroje zároveň a poslechnout si tak celou vytvořenou skladbu. Pro spuštění přehrávání hudby slouží

implementovaná metoda **play()**, ve které se spustí nové vlákno. Životní cyklus vlákna probíhá v implementované metodě **run()**. Zde se spustí metoda **playMusic(),** ve které se v cyklu postupně procházejí jednotlivé akordy z modelu a pro každý aktuální akord se projdou všechny tóny uvnitř akordu. Následně dochází k syntetizaci zvuku na midi kanále. Je nutné získat číslo tónu **NUM** a poslat na kanál zprávu **channel.NOTEON(NUM)**. V tu chvíli dojde pomocí midi syntetizátoru k vytvoření požadovaného tónu. V cyklu se tedy projdou všechny tóny z akordu, které tak tvoří souzvuk. Následně je nutné uspat vlákno na dobu, kdy má trvat znění těchto tónů. Tato hodnota se získá z tvaru noty a tempa skladby jednoduchým přepočtem na časovou hodnotu v milisekundách **LENGT**. Po probuzení vlákna je pak nutné zase znějící tóny vypnout. To se opět provede v cyklu pro všechny tóny v akordu a pomocí zaslání kanálové zprávy **channel.NOTEOFF(NUM)**. Po syntetizaci zvuku se všechny osnovy zobrazující přehrávané akordy posunou metodou **moveToNext()** na další aktuální akord.

Přehrávání hudební stopy se provádí tak dlouho dokud se nedojde na konec stopy, či nedojde k vyvolání metody **stop()**, která ukončí cyklus přehrávání akordů ještě před koncem. Tuto stopu je možné vypnout metodou **setMute**, tím se utlumí kanál pro syntetizaci hudby. Je také možné nastavit tuto stopu jako sólovou pomocí **setSolo**, to znamená, že bude hrát jen tato stopa ze všech stop nacházejících se ve skladbě, plus případné další sólové stopy. Před začátkem přehrávání stopy se vždy zjistí, jestli je možné stopu přehrát pomocí metody **checkChannelToPlay**. Pokud je stopa utlumena, nebo je nějaká ze stop sólová a tato není, pak nedojde ke spuštění vlákna pro přehrávání.

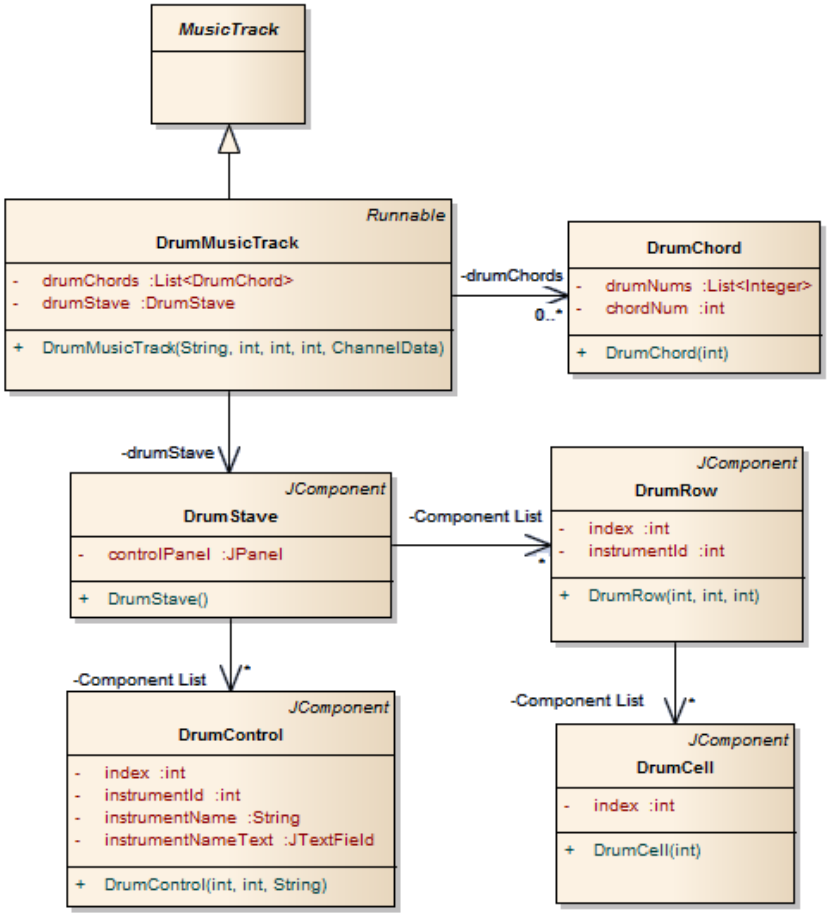
4.5 Bicí hudební stopa

Pro vylepšení funkcionality aplikace byl doplněn druhý typ hudební stopy a to stopa se záznamem úderů bicích nástrojů **DrumMusicTrack**. Je zde využíváno funkcionality posledního desátého midi kanálu, který poskytuje **Synthesizer** z midi systému. Tento kanál obsahuje od 35 kanálového indexu různé bicí a perkusivní nástroje, jako jsou bubny, činely, dřívka a další užitečné zvuky. Práce s tímto kanálem je v podstatě stejná jako je tomu o ostatních kanálů, které obsahují konkrétní tóny různých barev. Opět se používají kanálové zprávy `NOTEON` a `NOTEOFF` pro syntetizaci příslušného zvuku. Přehrávání hudební stopy se pak provádí obdobně jako tomu bylo u notové hudební stopy. Na obrázku *Obrázek 4.9* je vyznačen třídní diagram, propojení jednotlivých tříd.

Je nutné získat z bicí osnovy **DrumStave** data v seznamu bicích akordů **DrumChord**, které se pak v cyklu postupně syntetizují. Pro vytváření bicích akordů tedy slouží speciální bicí osnova. Ta je tvořena

čtvercovou maticí, kde každý řádek **`DrumRow`** představuje jeden bicí nástroj. Každý řádek pak má řídící komponentu DrumControl, ve které se nachází textové pole s názvem bicí nástroje a pomocí tažení je možné celý řádek přemístit na jinou pozici. Řádek nástroje je pak dále tvořen jednotlivými buňkami **`DrumCell`**. Ta je v základě nevyplněna, avšak vyplněním této buňky kliknutím myši zaznamenáváme, že příslušný bicí nástroj má udeřit v časovém úseku, daném pořadí buňky v řádku. Vyplněné buňky, které se nacházejí v jednom úseku nad sebou představují souzvuk nástrojů což je jeden bicí akord **`DrumChord`**. Pro orientaci v časových úsecích je zde nad buňkami komponenta **`DrumTacts`**, která vyznačuje jednotlivé časové úseky pomocí označení taktů.

Pro přehrávání bící hudební stopy se nejdříve musí získat seznam bící akordů **`DrumChord`** a to se provede v implementované metodě **`prepare()`**. Přehrávání se spustí implementovanou metodou **`play()`**, zde se vytvoří nové vlákno díky implementovanému rozhraní Runnable. Životní cyklus vlákna probíhá v metodě **`run()`**, v které se spustí metoda **`playMusic()`**. Zde se v cyklu postupně



Obrázek 4.9: Struktura bicí hudební stopy a jejích grafických komponent

projdou všechny akordy a syntetizují se zvuky všech nástrojů v jednotlivých akordech, tak jako v notové hudební stopě. Délka trvání každého akordu v milisekundách je dána hodnotou **oneBeatLength**, která je jednoduchým přepočten získána z tempa skladby.

4.6 Vlastní simulace interaktivních hudebních nástrojů

Pro plnohodnotnou funkci aplikace jsou zde naimplementovány komponenty, představující virtuální hudební nástroje, které lze ovládat myší (či klávesnicí) a simulovat tak hru na tyto nástroje. Jednotlivé nástroje jsou spojeny s konkrétními hudebními stopami. Díky tomu lze syntetizovat hudbu na Midi kanále vybrané stopy, či zvýrazňovat právě přehrávané tóny ve skladbě na vybraném nástroji. Tyto simulace hudebních nástrojů pak mohou hrát velkou roli při výuce hry na reálný hudební nástroj. Uživatel se lépe zorientuje, kde se zadaný tón/akord z notové osnovy zahraje na nástroji. Při skládání hudby je to také vhodná pomůcka, protože lze sestavit nějaký akord a poslechnout si jak zní i bez fyzického vlastnictví daného nástroje.

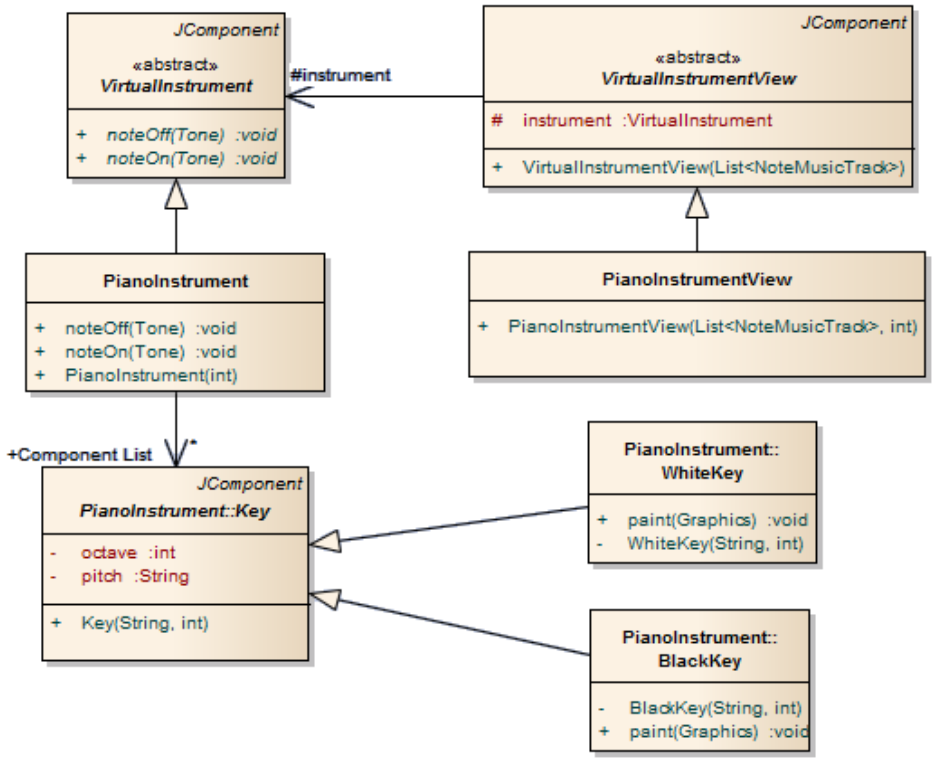
Abstraktní třída **VirtualInstrument** představuje virtuální hudební nástroj. Jsou zde dvě abstraktní metody a to **pressMarkedTones** a **releasedMarkedTones** sloužící pro zmáčknutí a puštění označených tónů na nástroji, tedy například u piána označené klávesy a u kytary označené pražce na vybraných strunách. Tyto dvě metody tedy jsou především pro hru akordů složených z označených tónů. Dále jsou zde dvě abstraktní metody **noteOn** a **noteOff** sloužící pro zmáčknutí vybraného tónu nástroje z externí komponenty. Díky tomu je možné při přehrávání notové osnovy zvýrazňovat hrané tóny na nástroji, bez zásahu uživatele.

Abstraktní třída **VirtualInstrumentView** slouží pro spojení grafického výstupu instrumentu VirtualInstrument s kontrolním panelem sloužícím k dodatečnému ovládání, či výběru hudební stopy spojené s tímto nástrojem. Nachází se zde instance z **InstrumentConnection** zajišťující propojení hudebního nástroje s vybranou hudební stopou. Dále obsahuje **musicTrackSelectionCombo** `JComboBox` sloužící pro výběr hudební stopy, která se v `InstrumentConnection` propojuje s instrumentem. Této komponentě se nastaví jako model pole hudebních stop MusicTrack. Dále je nutné pro správné zobrazovaní názvů hudebních stop v `JComboBox` nastavit vlastní **ListCellRenderer**, pomocí kterého se vykresluje renderovací objekt, který může být zastoupen jakoukoliv komponentou dědící z `JComponent`. V tomto případě je použit **JLabel**, kterému se nastaví text názvu stopy. Obsahuje tlačítko **markingBtn** pro nastavení na označování tónů v nástroji, dále tlačítko **pushBtn**, pomocí kterého se označené tóny stlačují či

pouštějí, tedy metody **`pressMarkedTones`** a **`releaseMarkedTones`**. Tlačítko **`channelBtn`** slouží pro zobrazení `ChannelView` komponenty, pro změnu vlastností kanálu vybrané hudební stopy.

4.6.1 Virtuální instrument piáno

Třída **`PianoInstrument`** představuje již konkrétní hudební nástroj a to tedy nástroj s klaviaturou - Piano. Tato klaviatura se skládá z bílých a černých kláves. Bílé klávesy představují celé tóny, a zleva doprava tóny *c, d, e, f, g, a, h*. Černé klávesy pak představují půltóny, které se nacházejí mezi jednotlivými celými tóny. Mezi klávesami celých tónů *e* a *f* se nenachází žádná klávesa půltónu, protože rozdíl mezi těmito celými tóny není jeden celý tón, jako je tomu u ostatních tónů, ale pouze půltón. To samé platí i pro tóny *h* a *c*, mezi kterými je taktéž rozdíl pouze půltónu. Sedm celých tónů a pět půltónů a tedy i jejich klávesy, představují tóny v jedné oktávě. Podle toho může být klaviatura rozdělena na části podle jednotlivých oktáv, kde nejnižší oktáva začíná zleva po nejvyšší vpravo. V jedné oktávě je tedy 7 bílých kláves a 5 černých kláves. Struktura komponenty je znázorněna na obrázku třídního diagramu *Obrázek 4.10*.

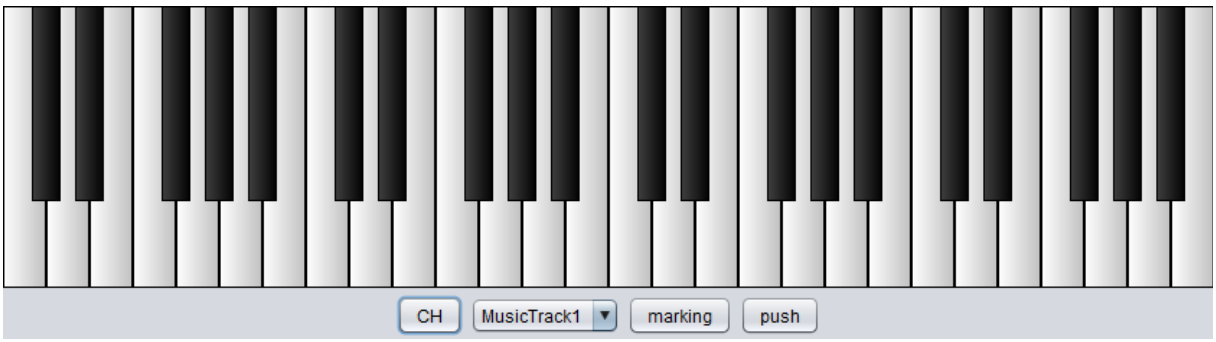


Obrázek 4.10: Třídní diagram struktury hudebního nástroje piáno

Abstraktní třída **Key** představuje klávesu. Parametr **pitch** určuje výšku tónu a hodnota parametru **octave** určuje oktávu ve, které se tón nachází. Tyto hodnoty slouží pro syntetizaci zvuku pomocí syntetizátoru a pro nastavení rozložení kláves v klaviatuře. Třída **WhiteKey** je již konkrétní bílá klávesa, která dědí z **Key**. Podle toho se také vykresluje. Třída **BlackKey** také dědí z `Key` a je to konkrétní černá klávesa klaviatury. Vykresluje se opět svým způsobem. Pomocí třídy **PianoLayout** se provede rozmístění jednotlivých kláves v komponentě piána. Bílé klávesy se podle výšky tónu a oktávy řadí vedle sebe zleva doprava. Černé klávesy se pak přiřazují na správná místa přes bílé klávesy, mezi které patří jejich půltón. Pro události vyvolané myší nad jednotlivými klávesami slouží

třída **PianoMouseListener**. Události, které se vyvolají nad klávesami zajišťují funkce jako jsou vstoupení a vystoupení kursoru myši z klávesy, podle toho se klávesy vykreslují. Důležitými událostmi jsou však `mousePressed` a `mouseReleased`, tedy zmáčknutí a uvolnění příslušné klávesy. Tím se vyvolají nové události **noteOn** a **noteOff**, které jsou odchyceny v připojených posluchačích a například v posluchači ve třídě spojení instrumentu s hudební stopou se syntetizují příslušné tóny pomocí kanálu hudební stopy.

Třída **PianoInstrumentView** dědí z `VirtualInstrumentView`. V konstruktoru se nastaví všechny potřebné parametry, předá se seznam hudebních stop, a vybere se aktuální hudební stopa, která se předá novému spojení instrumentu se stopou **InstrumentConnection**. Na obrázku *Obrázek 4.11* je znázorněno grafické rozhranní výsledné piáno komponenty.



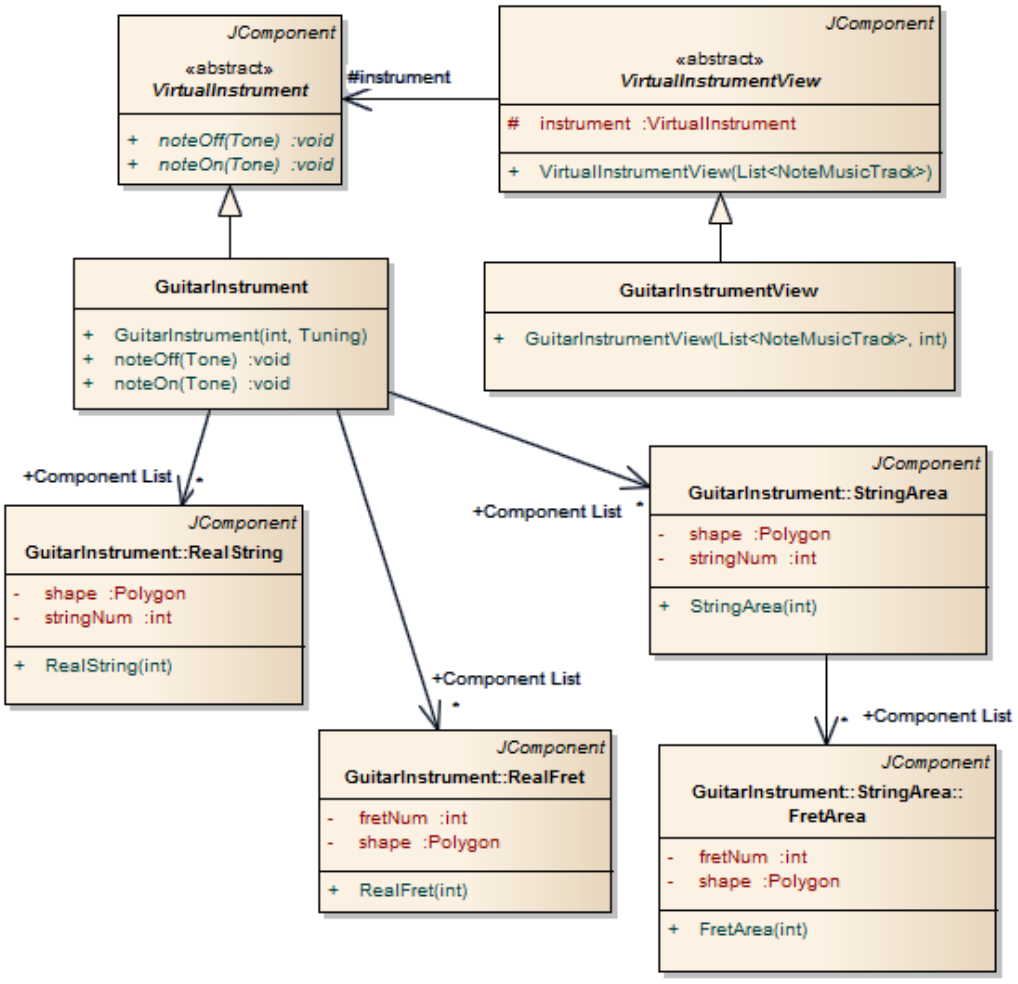
Obrázek 4.11: Grafické uživatelské rozhranní piánového nástroje

4.6.2 Virtuální kytarový nástroj

Třída **GuitarInstrument** dědí z `VirtualInstrument`. Představuje konkrétní kytarový nástroj. V tomto případě jsou možné dva kytarové nástroje a to klasická šestistrunná kytara a

čtyřstrunná basová kytara. Kytarový nástroj se vybere podle toho, jaký druh hudební stopy je s tímto nástrojem spojen. Pokud je hudební stopa v houslovém klíči, pak se zobrazí klasická kytara. Pokud je v basovém klíči, pak se zobrazí basová kytara. Grafická komponenta představuje pouze kytarový hmatník bez dalších grafických zobrazení kytary, protože pro funkci nejsou nutné. Jsou zobrazeny pouze jednotlivé struny a pražce. Celá komponenta je hierarchicky rozdělena do několika částí. Na obrázku *Obrázek 4.12* je zobrazen třídní diagram kytarového nástroje.

Hmatník je rozdělen nejdříve do jednotlivých horizontálních oblastí, které patří pod každou jednu strunu. Tyto oblasti představuje komponenta **`StringArea`**. Ta má své číslo **`stringNum`** podle toho pod kterou strunu tato oblast spadá. Obsahuje také svůj tvar **`shape`** z rozhranní **`Polygon`**, který určuje reálný tvar komponenty. **`StringArea`** se pak dále dělí na jednotlivé oblasti příslušející



Obrázek 4.12: Třídní diagram kytarového nástroje

pražcům a to **FretArea**. Tato komponenta má své číslo **fretNum** a má také svůj tvar. Dohromady všechny komponenty pak představují celý kytarový hmatník, kde každá oblast **FretArea** má svůj konkrétní tón.

Důležitým faktem je, že rozmístění jednotlivých pražců na kytarovém hmatníku není lineární a rozestupy nejsou vždy stejné, postupně se snižují směrem ke kytarové kobylce. Dále je nutné podotknout, že kytarový krk se od hlavy s nultým pražcem postupně ke kytarové kobylce rozšiřuje.

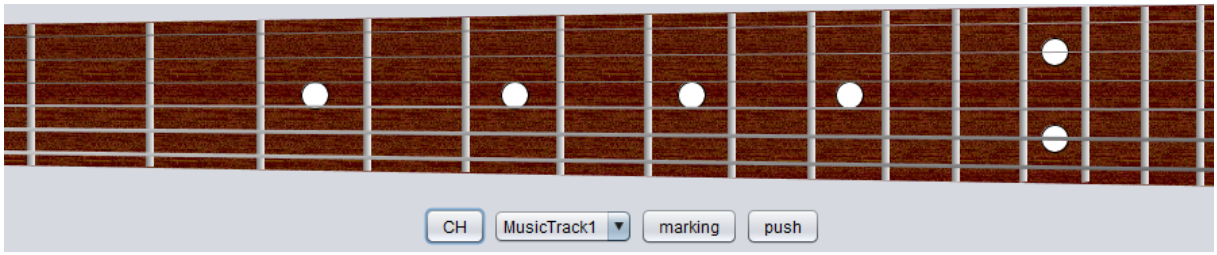
Z toho plyne, že tvary jednotlivých oblastí nebudou čtvercové a už vůbec ne stejné. Informace o rozestupu kytarových pražců byly čerpány ze zdroje **[19]**.

Poloha kytarového pražce je udána vztahem: L(n) = L / (k ^ n), Kde **n** je číslo pražce, pro který počítáme polohu, **L(n)** je vzdálenost pražce n od nultého pražce. **L** je délka celé struny, tedy od nultého pražce až po kytarovou kobylku. Hodnota **k** je koeficient odvozený od změny frekvence vibrace struny vůči její délce a napnutí struny. Tento koeficient má konstantní hodnotu 2 ^ (1/12) tedy asi 1.05946. Díky tomuto vzorci je možné vypočítat jednotlivé rozestupy pražců. Dále je nutné také vypočítat zkosený tvar jednotlivých oblastí. Výpočty se provádí nejdříve pro všechny struny v **GuitarLayout**. Zde se nastaví komponentám **StringArea** zkosený tvar podle délky hmatníku a výšky hmatníku na obou koncích. Dále jednotlivé tvary a pozice komponent **FretArea** se nastaví pomocí **StringAreaLayout**. Zde se již uplatní dříve zmíněný vzorec. Každé komponentě **FretArea** je přiřazen **MouseListener**, který slouží pro odchytávání událostí, které se vyvolají prací myši nad komponentou. Metodách MouseEntered a MouseExited se volají metody příslušné FretEntered a FretExited, pomocí kterých se nastaví jiná barva oblasti a tím se vyznačí oblast nad kterou se právě nachází kurzor myši. V metodách mousePressed a mouseReleased se volají přidružené metody pressFret a releaseFret tedy zmáčknutí a uvolnění pražce. Tím se vyvolají tak jako v instrumentu piána události **noteOn** a **noteOff**, které je možné opět odchytit v připojeném posluchači spojení instrumentu s hudební stopou InstrumentConnection, kde se pomocí předaných hodnot syntetizuje na kanále stopy příslušný tón.

Předcházející komponenty představovaly pouze kytarový hmatník rozložený na jednotlivé oblasti, s kterými je možno pracovat jako s tlačítky pro jednotlivé tóny. Je vhodné tedy, aby byly zobrazeny i skutečné kytarové pražce a struny. Skutečný kytarový pražec je reprezentován objektem ze třídy **RealFret**. Je graficky zobrazen jako čtyřúhelník s určitou šířkou. Podle jeho čísla je opět pomocí **GuitarLayout** umístěn na správné místo a je mu nastaven určitý tvar, tak aby zapadl na kytarový hmatník. Pokud se nachází kursor myši nad jednou z oblastí příslušející kytarovému pražci, je i tento

skutečný pražec barevně vyznačen. Pro lepší vzhled je tvar vybarven přechodem gradientu. To se provede v přepsané metodě JComponent **`paintComponent`**. Protože třída Graphics nepodporuje kreslení gradientu, je nutné ji přetypovat na instanci třídy Graphics2D. Pak je možno pomocí metody **`setPaint`** nastavit na kreslení gradientu předáním nové instance **`GradientPaint`** s nastavenými konkrétními parametry přechodu. Skutečná kytarová struna je reprezentování **`RealString`**. Ta má své číslo **`stringNum`** a podle něj se v **`GuitarLayout`** nastaví její pozice a tvar. Pokud se kurzor myši nachází v oblasti příslušné struny, pak se tato struna označí. Při vykreslování celé komponenty je nastaveno renderování grafiky na „antialiasing“, což umožňuje objekt třídy `Graphics2D`. To způsobí vyhlazení hran a tím pěknější vzhled.

Pro pěkný reálný vzhled je také načten obrázek textury dřeva a vytvořena instance **`TexturePaint`** s nastaveným obrázkem a čtvercovou plochou, která určuje velikost jednoho obrázku v textuře, který se pak dále pořád opakuje. Objekt z texturePaint je při vykreslování celého kytarového hmatníku nastaven jako výchozí vyplňující malba. Na obrázku *Obrázek 4.13* je vidět výsledné grafické rozhranní kytarové komponenty.

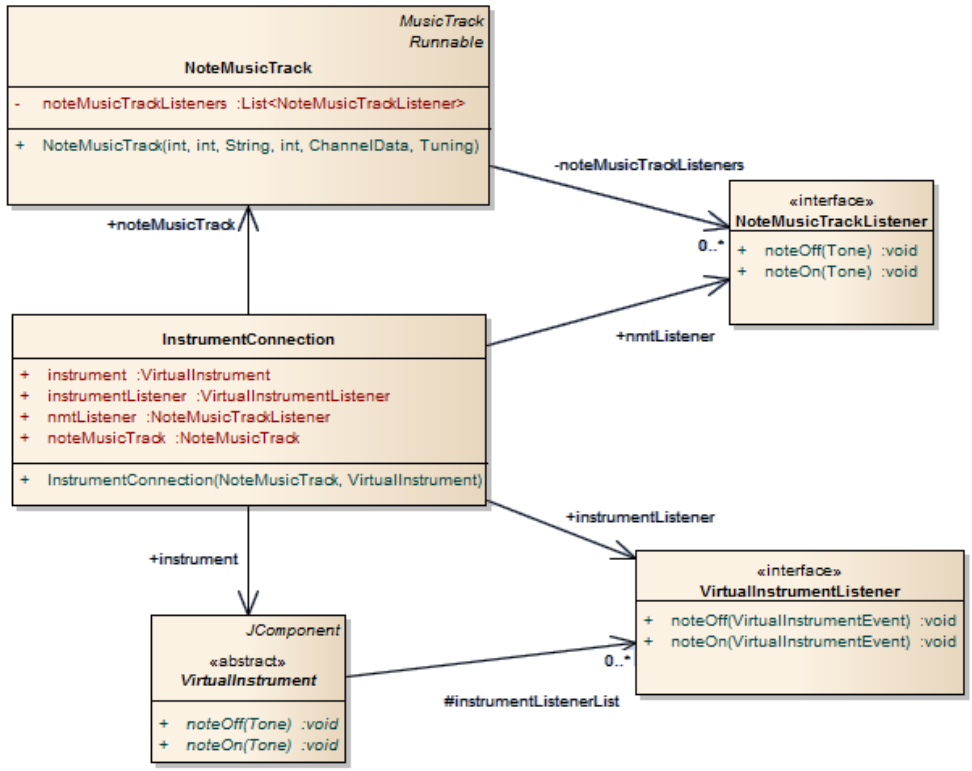


Obrázek 4.13: Grafické uživatelské rozhranní kytarového nástroje

4.6.3 Propojení virtuálního instrumentu s notovou hudební stopou

Každý virtuální instrument, který je v aplikaci vytvořen, ať už piáno, nebo kytara, je propojen s nějakou hudební stopou. Toto propojení je zajištěno pomocí třídy **`InstrumentConnection`**, která obsahuje jak virtuální instrument, tak samotnou hudební stopu **`NoteMusicTrack`**. Ta má připojen posluchač **`NoteMusicTrackListener`**, pro události vyvolané nad stopou. Těmito událostmi jsou zahrání určitého tónu v stopě a tento tón je skrze propojení vyznačen na připojeném nástroji. Lze i naopak poslouchat události nad virtuálním nástrojem pomocí `VirtualInstrumentListener`. Událost je pak vyvolána zmáčknutím klávesy na piánu, či pražce na kytarovém nástroji a předá se tón odpovídající klávese. Tento tón je pak skrze spojení

syntetizován na kanálu připojené hudební stopy. Celá struktura propojení je vyznačena na třídním diagramu v obrázku *Obrázek 4.14*.



Obrázek 4.14: Třídní diagram spojení virtuálního nástroje s hudební stopou

4.7 Doplňkové podpůrné nástroje

Pro doplnění funkcionality programu byly naimplementovány další podpůrné nástroje, které mohou nalézt uplatnění nejen při vytváření hudebních stop, ale i při výuce hry na nějaký nástroj.

4.7.1 Ladička hudebních nástrojů

Ladička je zařízení, které vydává tón s přesnou frekvencí, například ladící vidlice, vydávající tón A, nebo toto zařízení je schopno určit frekvenci znějícího tónu. Mimo jiné se ladička používá i pro měření a různé experimenty v akustice, nebo v medicíně.

Elektronická ladička měří frekvenci přiváděného tónu a podle toho určí výšku tónu ke kterému se blíží a rozdíl oproti tomuto přesnému tónu. Většinou jsou vybaveny ručičkovým ukazatelem, nebo

diodami určující odchylku měřeného tónu od přesného tónu. V dnešní době umí moderní elektronické ladičky mnoho funkcí, například mohou měřit frekvence ve velkém rozsahu, umožňují různé kalibrace, volbu mezi chromatickým měřením a měřením konkrétního tónu. Jsou také speciální ladičky určené výhradně pro konkrétní nástroj, na který jsou přizpůsobeny. Forma elektronické ladičky může být různá. Nejčastějším je příruční ladička s mikrofonem a linkovým vstupem, popřípadě výstupem. Ladička se však může vyskytovat i jako modul, který je součástí racku elektronického zesilovače hudebního aparátu, či může vypadat jako pedálový efekt zapojený do efektové smyčky s ostatními kytarovými efekty. Podle kvality zpracování, možnostech které ladička poskytuje a také formy zpracování ladičky se také odvíjí její cena, která není nízká. Další informace lze zjistit například z tohoto zdroje **[20]**.

Alternativou může být softwarová ladička v počítači. Na internetu je možné nalézt spoustu volně šířených ladiček, ať už na stažení, či fungujících on-line. Tyto volné ladičky však nemají tak dobré zpracování, avšak je možné nalézt relativně použitelné programy a pro potřeby obyčejného uživatele postačuje. Je možné si i zakoupit profesionální softwarovou ladičku, například jako samostatnou aplikaci, anebo jako *VST plugin*, který se může stát například součástí softwaru určeného pro stříhání a produkci hudby. Další možností je naprogramovat si vlastní ladičku. Na internetu je možné nalézt několik otevřených zdrojových kódů softwarové ladičky a to v různých programovacích jazycích od C++ až po Java. Pro moji aplikaci byl využit a upraven zdrojový kód z tohoto zdroje **[21]**

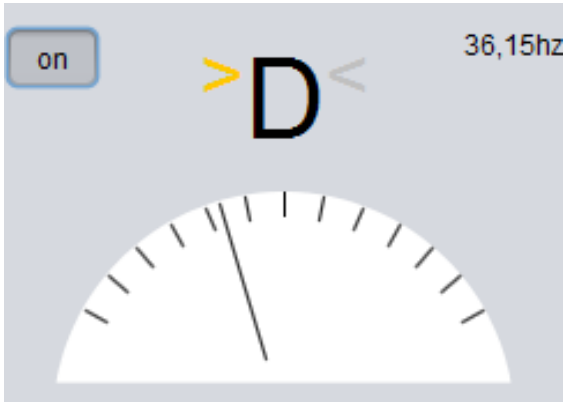
Ladička z tohoto kódu obsahuje i graf, který zobrazuje průběh vstupního signálu. Tento graf však není pro potřeby ladičky potřebný. Celkově byl upraven kód, tak aby mohla ladička fungovat jako další příslušenství v rámci aplikace.

Vlastní implementace je tvořena třídou **`GuitarTuner`** zajišťující samotnou funkci měření frekvence vstupního tónu a podle toho rozpoznávání nejbližšího tónu. Zobrazení grafického uživatelského rozhranní zajišťuje komponenta **`GuitarTunerView`**, jejíž instance je součástí třídy **`GuitarTuner`**. Třída má implementované rozhranní **`Runnable`**, díky čemuž je možné v implementované metodě **`run`** provádět životní cyklus vytvořeného vlákna. V této metodě je tedy obsažen celý algoritmus zjištění frekvence vstupního tónu a odvození názvu nejbližšího tónu. Pro zjištění frekvence ze vstupního vzorku se u mnoha ladiček používá tzv. **Rychlá Fourierova Transformace (Fast Fourier Transform)** což je algoritmus pro zjištění **Diskrétní Fourierovy Transformace** a její inverze. Ta převede diskrétní časově závislé hodnoty na frekvenčně závislé hodnoty a tím i frekvenci vzorku. U této ladičky byla však použita jednodušší metoda **autokorelační funkce**.

Nejdříve se vytvoří datový formát AudioFormat s určitými parametry pro čtení vstupních dat. Nejdůležitějším parametrem je `sampleRate`, který určuje počet vstupních vzorků za jednu sekundu, nastavený na hodnotu 44100. Následně se získá zdroj dat **`TargetDataLine`** z **`AudioSystem`**, pak se datová linka otevře a spustí. Vytvoří se pole bytů **`buffer`**. Následuje cyklus **`while`**, který probíhá dokud nenastane chyba při čtení vstupních dat z datové linky, nebo se neukončí vypnutím ladičky. V každém opakování se tedy metodou **`read`** z datové linky načtou hodnoty vstupních dat do pole **`buffer`**. V následujících krocích je nutné získat délku vzorku **`sampleLen`**, která určuje délku periody vstupního signálu. To se provede porovnáním jednotlivých vzorků z **`buffer`** a výpočtem diferencí mezi jednotlivými kroky. Rozsah se postupně zvětšuje a hledáním změny příznaku mezi předchozími a následujícími diferencemi se nalezne minimum funkce, tedy konec periody. Délku vzorků **`sampleLen`** pak určuje konečný rozsah vzorků, při kterém bylo nalezeno minimum funkce. Díky této délce periody a hodnotě **`sampleRate`** vstupních dat získáme podělením **`sampleLen/sampleRate`** výslednou **frekvenci** vstupních dat. Tato frekvence se následně znormalizuje tak aby byla v požadovaném intervalu, podle kterého určujeme výšku tónu. Frekvence se tedy buď násobí dvěma, tak dlouho dokud nepřekročí dolní hranici požadovaného rozsahu, nebo se naopak dělí dvěma tak dlouho dokud nepřekročí horní hranici rozsahu. Metodou **`closestNote`** se zjistí, který tón je nejblíže získané frekvenci a to tak, že projde statické pole **`FREQUENCIES`** obsahující přesné frekvence jednotlivých tónů a nalezne tón s nejmenším rozdílem. Ze statického řetězcového pole **`NAMES`** pak podle získaného indexu přesného tónu získáme název tónu. Ten se pak vypíše do grafického rozhranní. Následně je nutné zjistit odchylku získané frekvence od požadované. Díky rozsahu určeného požadovanou frekvencí a následující, či předešlé frekvence (podle toho jestli získaná frekvence je menší či větší než požadovaná), a díky odchylce je možné určit úhel ručičky ladícího měřiče v grafické komponentě.

Třída **`GuitarTunerView`** představuje grafické uživatelské rozhranní ladičky **`GuitarTuner`**. Tato komponenta obsahuje tlačítko **`startBtn`**, které po stlačení spustí vlákno v **`GuitarTuner`** a následně provádí uvedený algoritmus pro výpočet frekvence a blízkého tónu. To se provádí tak dlouho dokud se tlačítko opět nestlačí, čímž se ukončí cyklus zjišťování frekvence vstupního vzorku. Dále obsahuje `JLabel` **`freqLbl`** ve kterém se vypisuje frekvence vstupního tónu, pokud je nějaký změřen. `JLabel` **`matchToneLbl`** zobrazuje tón, který nejblíže odpovídá vstupnímu tónu. Odchylku od požadovaného tónu pak zobrazuje komponenta **`PendulumView`**, která v sobě vykresluje ručičku natočenou v určitém úhlu. Ten je tedy dán odchylkou vypočtenou v algoritmu. Pokud je ručička

natočena do svislé polohy v 90 stupních, pak je odchylka od požadované frekvence nejmenší, či nulová. K tomu se ještě navíc zobrazují dvě šipky, které zobrazují kterým směrem strunu ladit, aby došlo k naladění blízkého tónu. Pokud dojde k naladění velmi požadované frekvence s určitou malou odchylkou, pak jsou obě šipky zobrazeny zeleně a indikují tak správné naladění. Na obrázku *Obrázek 4.15* zobrazeno grafické rozhranní vytvořené komponenty ladičky.



Obrázek 4.15: Grafické rozhranní ladičky

Ladění hudebního nástroje může probíhat dvěma způsoby. Prvním je připojení mikrofonu do zvukové karty počítače. Zvuk z nástroje pak musí být tímto mikrofonem co nejlépe zachycen. Záleží i na kvalitě mikrofonu. Čím lepší mikrofon, tím lépe ladička rozezná frekvenci znějícího tónu. Pokud je však tón doprovázen příliš vysokým rušivým hlukem, či šumem, který prakticky nemá frekvenci, pak nelze určit vstupní frekvenci. Druhou možností je připojit hudební nástroj přímo do mikrofonního vstupu, to však platí pouze pro elektrické či elektronické hudební nástroje, jako je například elektrická kytara, elektrická baskytara, či jakýkoliv hudební nástroj s linkovým výstupem. V tomto případě by měla být kvalita snímaného zvuku mnohem větší než při snímání mikrofonem. Zde už jen záleží na kvalitě přívodních kabelů a případných přechodových adaptérů, které jsou nutné do 3.5 mm vstupu v případě 6.35 mm konektorů a nakonec i na kvalitě samotné zvukové karty. Může také docházet přebudění vstupního signálu příliš vysokou hlasitostí nástroje. Pak je nutné nastavit na zvukové kartě mikrofonnímu vstupu nižší úroveň zesílení.

Při zkoušení funkčnosti ladičky bylo využito elektrické baskytary, která byla zapojena přímo do mikrofonního vstupu. Funkčnost ladičky byla až nad očekávání dobrá. V porovnání s některými levnými ladičkami, se kterými jsem se setkal má velmi rychlou odezvu měření, což je logické vzhledem k rychlosti výpočtu na počítači oproti čipu v elektronické ladičce. Výhodou je také to, že na

rozdíl od většiny ladiček ukazuje i přesnou frekvenci tónu, takže nemusí sloužit pouze pro ladění nástroje, ale i zjišťování přesné frekvence nějakého zvuku. Nevýhodou je však nutnosti počítače a z toho vyplývající nemobilnosti. Řešením by bylo například implementace ladičky pro nějaké mobilní zařízení, čímž se může taková ladička stát zajímavou doplňkovou aplikací, avšak zkušený hudebník raději zvolí profesionální ladičku, která svůj účel splní lépe. V rámci hudební aplikace však tato implementace může hrát významnou roli, pokud uživatel například skládá hudební skladbu, či se učí hudební skladbu za pomoci kytarového nástroje, který se delším hraním může rozladit.

4.7.2 Metronom

Dalším užitečným nástrojem v aplikaci je vlastní implementace metronomu. Metronom je mechanický, elektromechanický, nebo elektronický přístroj, který rovnoměrně přesně odklepává rytmus hudební skladby. V základním nastavení pak jeden úder metronomu představuje jednu čtvrťovou notu. Od toho se i odvíjí tempo metronomu, což je nastavitelný parametr, udávající rychlost rytmu, většinou se udává v úderech za minutu. Z anglického názvu se pak udává hodnota tempa v jednotkách *bpm*, tedy **beats per minute**. Navíc metronom většinou obsahuje zařízení, které zvýrazní například druhý, třetí, čtvrtý, či šestý úder, jako „těžkou dobu“, což označuje vždy počátek každého taktu. Hodnoty tempa se tak pohybují mezi 40 až 210 *bpm*, kde hodnota 40 je nejnižší tempo a odpovídá velmi pomalým skladbám a hodnota 210 je nejvyšší tempo, které odpovídá nejsvižnějším skladbám. Mimo tyto hodnoty se prakticky nezachází z hlediska využitelnosti a poslouchatelnosti skladeb. Navíc není nutné tempo nastavovat mimo tyto hranice, protože vyšší i nižší tempa lze odvodit z temp v intervalu násobením či dělením dvěma. Mechanický metronom má ručičku, kterou ovládá hodinový stroj s pružinou a reverzním kyvadlem. Rychlost tempa se pak ovládá závažíčkem umístěným na ručičce. Elektromechanické mají motorek, či jiné zařízení. Ručička metronomu se pohybuje z jedné strany na druhou a při doražení na jednu stranu vydá zvuk označující jeden úder. Elektronické metronomy mohou mít několik různých forem. Mohou například vydávat jen zvuk úderů, či obsahovat sadu LED naznačující jednotlivé údery bliknutím, či přímo obsahují display, na kterém je zobrazena pohybující se ručička metronomu. Tak jako u kytarových ladiček se i u metronomů odvíjí kvalita od ceny. Mechanické metronomy byly vždy velmi nákladné, protože se jednalo o velmi přesné a jemné mechanické strojky. Elektromechanické a elektronické metronomy se staly později více dostupné. Často bývají metronomy součástí lepších ladiček, či jiných hudebních zařízeních, jako je elektronický bubeník apod. Další informace o metronomu lze nálezt ve zdroji **[22]**.

Implementaci tvoří třída **`Metronom`**. Ta představuje funkční část metronomu. Obsahuje instanci třídy **`MetronomView`** sloužící jako grafické uživatelské prostředí. Metronomu je nutné při inicializaci

předat instanci kanálových dat **ChannelData**, které slouží pro syntetizování zvuku jednotlivých úderů. Pro potřeby metronomu je nutné získat z **MidiSyntehsizer** poslední desátý kanál, který je od indexu 35 rezervován pro bicí a perkusivní nástroje. Je implementováno rozhranní **Runnable**, díky kterému je možné pracovat s třídou jako s vláknem, kde životní cyklus vlákna probíhá v implementované metodě **run()**. Zde se v cyklu vždy na kanále zavolá metoda `NOTEON` s příslušným indexovým číslem tónu, který se má syntetizovat. Poté se vlákno uspí na přesnou dobu uloženou v proměnné **oneBeatLength**, což je délka jednoho úderu v milisekundách. Tato hodnota je vypočtena z nastavené proměnné **tempo**. Tempo se dále může měnit ve vymezeném intervalu a tím zrychlovat a zpomalovat metronom. Po probuzení vlákna se zavolá na kanál metoda **NOTEOFF** opět s příslušným číslem tónu. V časovém úseku mezi metodami `NOTEON` a `NOTEOFF` se syntetizuje zvuk tónu, v konkrétním případě desátého kanálu pouze úder perkusivního nástroje. Dále se zde rozlišuje, kolikátý je aktuální úder v taktu a tím se vždy na první dobu syntetizuje odlišný zvuk úderu oproti ostatním, aby tuto dobu zvýraznil – těžká doba.

Metronom dále obsahuje ovládání kyvadla metronomu. Pro to slouží instance třídy Pendulum, která také implementuje rozhranní **Runnable**. V metodě **run()** se provádí nastavení kyvadla a to tak, že se vždy mezi jednotlivými údery v čase `oneBeatLength` po jednotlivých krocích přičítá či odčítá úhel, který svírá kyvadlo s vodorovnou osou. Vždy když kyvadlo dorazí na maximální vychýlení, současně se provede jeden úder metronomu a následně se otočí směr dalšího kyvu. Každý krok změny úhlu se vždy projeví na grafickém výstupu.

Třída **MetronomView** slouží jako grafické uživatelské rozhranní metronomu. Obsahuje tlačítko **startBtn**, které spustí běh metronomu metodou **startMetronom()**, ve kterém se spustí obě vlákna, pro syntetizování zvuku metronomu a pro ovládání grafického kyvadla metronomu. Tlačítko **stopBtn** pak obě vlákna zastaví a tím i běh metronomu. Pomocí komponenty **tempoSpinner** je možné nastavovat tempo metronomu a to i za běhu. Dále komponenta obsahuje **PendulumView**, což je komponenta zobrazující kyvadlo metronomu. To je vykresleno jednoduše jako linka, která může mít úhel náklonu v určitém rozmezí. Úhel náklonu pak řídí vlákno **Pendulum**, a tím i průběh kyvadla.

Metronom je užitečný především při hře na nějaký nástroj, protože odklepává přesný rytmus ve stejném tempu. Hráči na nástroj pomáhá se držet tohoto rytmu a umožňuje hrát přesněji. Metronom tak nahrazuje například bubeníka, který by měl také hrát přesně, při hře s kapelou.

4.8 Pomocné třídy

Pro usnadnění práce bylo naimplementováno několik pomocných tříd.

4.8.1 Třída MusicBox

Třída **MusicBox** sdružuje různé hudební konstanty jako jsou seznamy výšek tónů apod., které jsou používány v celé aplikaci. Dále obsahuje velmi důležité převodové funkce mezi jednotlivými hudebními datovými strukturami. Má například nadefinovaný seznam všech sedmi tónů v poli PITCHES. Také obsahuje dvě základní lazení ze třídy Tuning pro kytaru i pro baskytaru. Pro virtuální kytarové instrumenty obsahuje také seznam reálných tlouštěk kytarových a baskytarových strun, pro docílení přesného vhledu. Také obsahuje parametr **FRET_SPACING_CONSTANT** což je důležitá konstanta, díky které je možné vypočítávat rozložení kytarových pražců na kytarovém nástroji. Dále obsahuje základní, maximální a minimální hodnoty tempa, které skladby mohou mít. Metoda **getMidiLength** metoda vrací hodnotu délky tónu, tedy jak dlouho má znít, v milisekundách. Tato hodnota je vypočtena pomocí parametrů **noteShape**, což určuje jak velkou část zabírá délka tónu jeden takt. Hodnoty parametru mohou nabývat. Parametr **tempo** určuje počet úderů noty čtvrťové za jednu minutu (bpm). Díky metodě **getMidiNum** je možné spočíst index tónu v midi kanále a to pomocí parametrů výšky tónu **pitch**, posunku tónu o nějaký půltón **accid** a pomocí oktávy **octave**, ve které se tón nachází. Spočtená hodnota se využije pro získání konkrétního tónu v kanále, který se nachází ve zvukové bance MidiSystemu.

Dále jsou uvedeny důležité metody pro hudební konverze. Metoda **getTone** provádí převod z midi tónu určeného číslem **midiNum** na standardní tón. Je nutné znát také tvar noty **noteShape** a jednoznačné **id** tónu. Parametr **type** určuje, zda se výšky tónů budou vybírat z řetězců **NORMAL_PITCH_ACCID**, či **DROPPED_PITCH_ACCID**, nebo **RAISED_PITCH_ACCID**. Parametr **keyType** se nastaví samotnému tónu, pro rozlišení ve, kterém klíči je nota zapsána a tím se uvnitř **Tone** dopočte hodnota **lineNum** čísla linky, na kterou se má daná nota posadit. Metoda **getTab** provádí pomocí indexu tónu v midi kanále a příslušného ladění kytarového nástroje převod na odpovídající kytarový **Tab**. Dále je potřebný tvar noty **noteShape** a unikátní **id**. Důležitým faktem je, že každý tón má svoji frekvenci a je tím tak unikátní. Zde je sice tón zastoupen indexem v midi kanále, avšak pod tímto indexem je umístěn právě tón s touto frekvencí. Avšak v rámci jednoho ladění se může jeden konkrétní tón nacházet na více místech kytarového hmatníku, například tón E

v oktávě 4 je možné ve standardním ladění zahrát na nejhlubší struně číslo 1. (struna E) a to na 12tém pražci. Tento samý tón však lze získat rozezněním struny číslo 2. (struna A) na 7. pražci hmatníku. Obě polohy mají po rozeznění struny stejný kmitočet, tím tedy i stejný tón. Z toho vyplívá, že převod není jednoznačný a pro každý tón může existovat dva a více tabů vzhledem k počtu strun kytarového nástroje a jeho ladění. Zde hraje velkou roli předaný parametr **`type`**, který určuje, jak má algoritmus při převodu postupovat.

Postup je následující:

1. V cyklu se projde celý objekt **tuning**, tedy všechny naladěné tóny všech strun nástroje, pro který chceme vytvořit tab.
2. Získají se hodnoty výšky tónu **pitch** a oktávy **octave** a díky metodě **getMidiNum** se získá výška tónu struny určená midi číslem **midiNum2**
3. Pokud je toto číslo **midiNum2** větší nebo rovno předloženému **midiNum**, znamená to, že tab ekvivalentní k tomuto tónu se nachází na této struně. Dále je nutné vyhovět podmínce aby rozdíl těchto dvou midi čísel nepřesáhlo počet možných pražců na kytaře dané konstantou **MAX_FRET_NUM**.Pokud se vyhoví podmínkám, pak se do seznamu **stringEqs** uloží číslo struny, na které se tab nachází a do seznamu **fretEqs** se uloží číslo pražce, které je se vypočte pouze odečtením **midiNum2** struny od požadovaného **midiNum**. To vyplívá z toho, že mezi dvěma pražci je rozdíl tónů jeden půltón a u midi indexů, je tomu tak stejně.
4. Podle velikosti seznamu ekvivalentních tabů a podle typu se vybere další vylučující se postupy :
    a. `TAB_LOWEST_STRING` – vybere se z ekvivalentních seznamů tab s nejhlubší strunou.
    b. `TAB_MIDDLE_STRING` – pokud je více jako jeden ekvivalentní tab, pak se vybere tab s druhou nejhlubší strunou.
    c. `TAB_HIGHEST_STRING` – pokud se vyskytuje více ekvivalentních tabů, pak se vybere ten, který se nachází na nejvyšší struně.
    d. `TAB_SHORTEST_DISTANCE` – provede výběr z ekvivalentních tabů ten tab, který má číslo pražce mezi určitými hranicemi, které jsou určeny rozdíly počtů půltónů mezi jednotlivými strunami. Tyto rozdíly jsou vyhodnoceny pomocí metody **getTuningDeltas**, čímž se získá pole hodnot. Delta pro první strunu je 0, ostatní jsou

odvozeny od rozdílů midi indexů ladění. Dolní hranice pro výběr pražce je delta pro aktuální strunu. Horní hranice je pak součet delt aktuální a následující struny. Pro poslední nejvyšší strunu pak není určena žádná horní hranice.

Tento postup nám zajistí to, že vybraný tab má přirozenější umístění, naproti ostatním tabům, které jsou většinou na příliš vysoké struně ale na nízkém pražci, či naopak na příliš vysokém pražci a nízké struně. Výhodou je, že tento postup se přizpůsobí i zvolenému ladění strun.

5. Vybraný postup vybere ze seznamu ekvivalentních tabů jeden a vrátí se jeho nová instance s příslušnými hodnotami.

Metoda **`getToneByTab`** provádí přímý jednoznačný převod z Tabů na Tóny pomocí ladění tabů **`Tuning`**. K převodu se využije předešlých metod. Nejdříve se zjistí hodnota midi čísla z tab a ladění tuning metodou **`getMidiNum`**, což nám dá jednoznačné určení tónu. Následně se provede převod z midi čísla na tón metodou **getTone**, typ převodu se zvolí pro potřeby projektu **`NORMAL`**, takže se vytvoří tón v normálním nesníženém ani nesníženém zápise. Je také potřeba určit, v jakém notovém klíči se bude nota zapisovat pomocí **`keyType`**, což pak určí číslo linky v notové osnově, na kterou se má posadit.

4.8.2 Ostatní třídy určené pro statický přístup

`ResourceManager` slouží k přístupu ke zdrojům, potřebným především pro vzhled aplikace. Tyto zdroje jsou uloženy v adresáři **resources**. Typickými zdroji jsou různé hodnoty či texty tlačítek, oken, labelů apod. Dále pak obrázkové ikony, které se umisťují do tlačítek, menu, nebo například Tray ikona aplikace, či obrázek pro tzv. splash screen.

`IOManager` slouží pro ukládání a exportování dat do souboru, či získávání dat ze souborů. Typickými úkony jsou uložení, či otevření projektu, export do tabulaturového formátu, export do xml, kde se využije další třída MusicXmlManager, export do **.mid** souboru apod.

`MusicXmlManager` je určen pro vytváření nových xml dokumentů z dat aplikace, či načítání dat z XML dokumentů do aplikace. Pro práci s XML bylo využito tříd z **`org.w3c.dom.*` API**. Slouží především pro uložení celého projektu do xml dokumentu, či načtení celého projektu z dokumentu. Také slouží pro export vybraných stop projektu do standardního formátu **MusicXML**.

TrayIconManager funguje jako `Singleton`. Slouží pro práci se **SystemTray** a **TrayIcon** **java.awt.* API**. SystemTray představuje např. pro Windows oznamovací část v hlavním panelu. TrayIcon pak představuje ikonu, kterou lze umístit do prostoru SystemTray. Lze nastavit obrázek a dále přiřadit vyskakovaní menu, které se zobrazí při kliknutí pravým tlačítkem na ikonu. V tomto menu jsou umístěny položky, pomocí kterých je možné aplikaci určitým způsobem ovládat. Je také možné zasílat různé oznamovací, či výstražné a chybové zprávy na tuto ikonu. Tyto zprávy pak se pak zobrazují u ikony v hlavním panelu.

4.9 Export a import standardních formátů

Celý hudební projekt je možné uložit do souboru. Ten je strukturován pomocí XML jazyka. V souboru jsou zaznamenány všechny důležité informace, jako název projektu, nastavení tempa a taktů a pak nastavení jednotlivých stop, včetně nastavení jejich kanálových stop. Nejdůležitějšími jsou však hudební data. U notových hudebních stop jsou uloženy všechny akordy a jejich tóny. U bicí stopy jsou uloženy všechny údery. Projekt se ukládá položkou **Save**, či **Save As..** v hlavním menu File. Pro vytvoření nového XML dokumentu se využívá třídy **MusicXmlManager.** Projekt je pak ukládán skrze třídu **IOManager** do souboru s vlastní příponou ***.mol**. Tento uložený projekt, je možné opět načíst položkou **Open** v menu **File**. Pokud je otevřený nějaký jiný projekt, pak se ten zavře a pomocí `MusicXmlManager` se rozprarsruje XML dokument přečtený ze souboru pomocí `IOManager` a vytvoří se z něj nový projekt se všemi daty a nastaveními, které byly v souboru uloženy.

Další možností, jak uložit svůj projekt, je export hudebních dat do hudebního souboru, v tomto případě do MIDI stopy. Nejdříve je však nutné získat sekvenci hudebních dat `javax.sound.midi.Sequence`. To lze dvěma způsoby. Vždy se však využívá třídy MidiRecorder. Prvním způsobem je využití grafického rozhranní rekordéru `MidiRecorderView`, kde tlačítkem **Rec** se spustí nahrávání. Do sekvence se pak zaznamenávají všechny MIDI zprávy, které jsou vyvolány v rámci všech kanálů. Lze tedy zaznamenat i hru na samotný virtuální nástroj, nebo i například tikání metronomu, protože i ten vytváří zvuk pomocí systémového MIDI zařízení. Také je možné spustit nahrávání z nástrojového panelu, tlačítkem pro záznam `recBtn` (červený bod). Tím se také spustí nahrávání, s tím že se ve stejný okamžik spustí přehrávání všech hudebních stop. Nahrávání se ukončí při dohrání všech zapnutých stop a zobrazí se komponenta `MidiRecorderView`, kde se ukáže v seznamu právě zaznamenaná sekvence. Tu lze pak uložit do souboru. K tomu se využívá IOManager, který sekvenci zapíše do souboru s příponou ***.mid**.

Druhým způsobem je využití položky v menu File->Export As..->Midi File, kdy se zobrazí okno MidiExportChooser, ve kterém je možné vybrat stopy, které chceme zahrnout do jednoho hudebního souboru. Tlačítkem Export Selected se vybrané hudební stopy zaznamenají pomocí metody **`exportToMidi`** z `MidiRecorder` do nové sekvence, která se pak opět za pomoci IOManger uloží do souboru ***.mid**.

Je také možné exportovat hudební notové stopy ve tvaru kytarové tabulatury tzv. *ASCII tab*, což je textový souborový formát, pomocí kterého je možné tabulaturu reprezentovat jako čísla, písmena a znaky ASCII. To lze provést skrze nabídku File->Export As..-> ASCII Tabulature. Zobrazí se seznam všechno notových hudebních stop, které je možné převést do tabulatury. Vybranou stopu pak exportujeme tlačítkem **Export**. Tím se pomocí `IOManager` vygeneruje textový soubor s vytvořenou ASCII tabulaturou a uloží se. Je možné následně vytvořený textový dokument otevřít přímo z aplikace v jeho asociovaném programu pro spuštění. To umožňuje třída **`java.awt.Desktop`**, která jej otevře pomocí metody **`open`**.

Poslední možností exportu hudebních dat, je pomocí **MusicXML**. Je to standardní formát pro digitální záznam hudby a distribuci, který je zapsán v souboru pomocí XML jazyka. Informace o MusicXML byly čerpány ze zdroje: **[23]**. Měl by sloužit jako univerzální formát pro západní hudební notaci. Hudební informace jsou pak navrhnuty tak, aby je bylo možné používat například programy pro notaci, sekvencery a další programy pro výuku a pro hudební databáze. Tento formát umožňuje uložení nejen hudebních dat, ale také uložení vzhledu notového zápisu jako jsou snížení a zvýšení jednotlivých not a další nastavení, které není možné uložit například do MIDI formátu, který ukládá pouze výšku tónu, nebere však třeba ohled nad rozdílem not F# a Gb, které sice mají stejnou výšku, ale v notové osnově jsou zapsány jinak. Data jsou hierarchicky uspořádána podle jednotlivých částí, představujících nástroje pro skladbu. Každá část má svoje nastavení včetně notového klíče a dalších atributů a následně již hudební data, představujících jednotlivé noty a jejich vlastnosti, jako jsou výška noty a délka noty. Dalšími možnosti dat, které je možné uložit je například zápis doprovodného zpěvu, nebo celkového vzhledu notového zápisu a dalších doplňkové informace. V aplikaci bylo však využito především uložení jednotlivých notových hudebních stop s jejich notami.

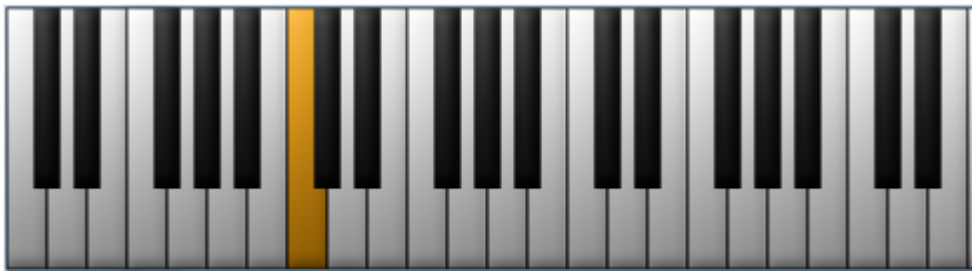
5 Implementace doplňkové aplikace pomocí JavaFX

Pro demonstraci funkcionality, kterou JavaFX poskytuje pro vývoj desktopových aplikací byla naimplementována aplikace **VirtualPianoFX**. Představuje tak analogii ke komponentě **`PianoInstrument`** v aplikaci MusicOnLine, tedy virtuálního hudebního klaviaturového nástroje piáno, aby bylo možné srovnat implementaci pomocí komponent JavaFX a Swing komponent.

5.1 Struktura JavaFX aplikace

Pomocí vývojového prostředí NetBeans 7.1 byl vytvořen nový projekt jako JavaFX Application. Tím se vygenerovala nová hlavní třída, která byla pojmenování VirtualPianoFX, která dědí z `javafx.applicationAppliaction`. Tato třída tedy tvoří samotnou aplikaci. Aplikace se spustí z hlavní spouštěcí metody pomocí statické metody **`launch`**. Aplikace má implementovanou metodu **`start`**, která se volá při spuštění aplikace. Předávaná instance **`Stage`** je kontejner na nejvyšší úrovni a je konstruována přímo platformou. Představuje tedy hlavní okno aplikace, do kterého je možné vložit scénu s dalšími grafickými komponentami. V metodě **`init`** se tedy vytvoří scéna aplikace a přiřadí se hlavní zobrazovaný kontejner **`primaryStage`**. Tak jako u komponenty **`PianoInstrument`**, se zde skládá scéna z jednotlivých kláves piána, které mohou být černé a bílé. Abstraktní klávesu představuje třída **`Key`**, která dědí ze třídy **`Rectangle`**, což je tvar čtverce. Klávesa dále nese číslo tónu klávesy a označení klávesy hodnotami **`step`** a **`octave`**, pomocí kterého lze identifikovat a umístit tuto klávesu. Konkrétní bílou klávesu pak reprezentuje třída **`WhiteKey`** a černou klávesu **`BlackKey`**. Podle toho jestli je klávesa zmáčknuta, nebo jestli se kurzor myši nachází nad klávesou, se vykresluje. Při vygenerování klaviatury v metodě **`generateKeyboard`** se vytvoří všechny klávesy, kterým se přiřadí jejich označení a také příslušné posluchače událostí pro interakci s myší. Pro každou událost se přiřadí jeden posluchač. Pro rozvržení kláves v klávesnici slouží metoda **`layoutKeys()`**, ta rozmístí klávesy podle jejich parametrů. Projekt obsahuje třídy `MidiSynthesizer` a `ChannelData`, které byly naimplementovány pro projekt MusicOnLine. Zde budou také sloužit pro syntetizaci zvuku na kanále. To se provede vždy vyvolání události **`MousePressed`** nad jednou z kláves. Zde se v implementované metodě **`handle`** zpracuje událost, označí se klávesa jako zmáčknutá a na kanále se vyvolá metoda **`NOTEON`** s příslušným kanálovým číslem, které klávesa obsahuje. Při uvolnění klávesy se vyvolá událost **`MouseReleased`**, ve které se klávesa odznačí a na midi kanál se pošle metodou `NOTEOFF` s příslušným parametrem čísla tónu,

zpráva o vypnutí znějícího tónu. Na obrázku *Obrázek 5.1* je zobrazeno výsledné grafické uživatelské rozhranní aplikace.



Obrázek 5.1: Výsledné grafické rozhranní JavaFX aplikace

5.2 Srovnání se Swing

Postup vytváření aplikace je velmi podobný k aplikacím založených na Swing komponentách. V JavaFX byla však převzata kontrola nad vytvářením samotné aplikace. Některé funkce jsou více zjednodušené. Layoutování komponent se dá provádět pomocí zabudovaných Layout Panes. Chybí tu však nějaký nástroj pro vlastní nastavení zarovnání komponent, jako je tomu u Swing s **`LayoutManager`**. Výhodou je jednoduššího nastavení různých efektů, které lze scéně a jednotlivým prvkům scény přiřadit. Jsou to například osvětlení, zrcadlení a další. Docílí se tak velmi pěkné grafiky jednoduchou cestou, než je tomu u Swing. Obsahuje další spoustu různých vestavěných funkcí, které slouží pro zjednodušení práce s grafikou. Například funkce sledování křivky objektem, které může sloužit pro efektní animace. Také umožňuje jednoduchou práci s objekty pomocí transformací a translací a lze tak v relativně jednoduchým způsobem vytvářet 3D scénu. Grafika je vykreslována pomocí nejnovějších technologií díky enginu Prism, který umožňuje hardwarovou i softwarovou akcelerace. Obsahuje sadu kontrolních grafických prvků, jako jsou tlačítka apod. a díky tomu je možné vytvářet klasickou desktopovou aplikaci. Celkově je práce při vytváření aplikace jednodušší, protože mnoho úkonů, které je potřeba provést již zajišťuje samotné JavaFX. Tím se sice více ztrácí kontrola nad správou aplikace.

6 Závěr

Výsledkem práce je funkční program využívající Java technologií pro tvorbu desktopových aplikací. Program slouží především pro tvorbu hudebních skladeb, které mohou být tvořeny několika stopami, představující různé hudební nástroje. Na stopy je možné pohlížet pomocí několika možných náhledů, především pomocí notové osnovy, ale také pomocí dalších alternativních náhledů jako je kytarový zápis tabulatury, nebo vlastní tabulatura. Celou hudební skladbu je možné přehrát a syntetizovat tak zvuk jednotlivých stop. Stopy je možné vypínat, či nastavovat jako sólové. Lze každé stopě nastavit barvu hudby výběrem hudebního nástroje, který má představovat. Při přehrávání se zvýrazňují v příslušných náhledech právě hrané tóny/akordy, což může sloužit jako dobrá pomůcka například při výuce na nějaký hudební nástroj. K tomu slouží i virtuální hudební nástroje, které je možné spojit s hudebními stopami. Nástroje umožňují interaktivní práci, takže je možné například na kytaře zahrát vyznačený akord. Při přehrávání hudební skladby se také na nástrojích zvýrazňují právě hrané akordy, takže je možné si lépe představit kde se daný akord hraje na příslušném nástroji. Výslednou hudební stopu je možné importovat a exportovat do různých formátů včetně standartního formátu MusicXML. K dispozici jsou další podpůrné nástroje, jako je například ladička hudebních nástrojů a metronom. Práce splnila všechny specifikace zadání. Program byl doplněn o již zmíněné virtuální nástroje a o samostatnou aplikaci vytvořenou pomocí JavaFX platformy. Při implementaci bylo vytvořeno mnoho vlastních datových struktur, včetně komponent pro grafické uživatelské rozhranní, ale také pro usnadnění přístupů v rámci celé aplikace. Dále také byly vytvořeny vlastní algoritmy, například pro převod mezi notovými zápisy a alternativními kytarovými tabulaturami. Díky předpřipraveným abstraktním třídám je možné budoucím rozšířením o další virtuální nástrojů. Také je možné doplnit další alternativní pohledy na notovou osnovu zkonstruováním vlastní sady obrázků alternativních tónů. Aplikace by mohla získat uplatnění hlavně u začínajících hudebníků, protože poskytuje jednoduchý způsob, jak zaznamenat svoji skladbu a díky tomu se zdokonalovat nejen v hudební teorii, ale také ve hře na hudební nástroj.

7 Použitá literatura

[1] : Luděk Zenkl, ABC hudební nauky, 2003ISBN 80-86385-21-3
[2] : Wikipedia.org, srovnání kytarové tabulatury s notovým zápisem, URL: <http://en.wikipedia.org/wiki/File:Diatonic_scale_on_C_tablature_clef.png>
[3] : Ladisla Daniel, Škola hry na altovou zobcovou flétnu, ISBN 35-072-83
[4] : Java SE Desktop Overview, URL: <http://java.sun.com/javase/technologies/desktop/>
[5] : java.net - Java Community server, Java desktop, URL: <http://community.java.net/javadesktop/>
[6] : Wikipedia.org - Java Swing, URL: <http://en.wikipedia.org/wiki/Swing_%28Java%29>
[7] : Oracle Documens - Java SE - Swing, URL: <http://docs.oracle.com/javase/6/docs/technotes/guides/swing/>
[8] : JDesktop Integration Components -javadesktop.org, URL: <http://javadesktop.org/articles/jdic/index.html>
[9] : JDesktop Integration Components - java.net, URL: <http://java.net/projects/jdic>
[10] : Introducing the JDesktop Integration Components, URL: <http://www.ibm.com/developerworks/java/library/j-jdic/>
[11] : JDIC - Java SE Integration , URL: <http://java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/desktop_api/>
[12] : Swing Application Framework - Swinglabs project, URL: <http://java.net/projects/appframework>
[13] : Swing Application Framework in NetBeans, URL: <http://netbeans.org/kb/68/java/gui-saf.html>
[14] : Swinglabs Projects on java.net, URL: <http://java.net/projects/swinglabs>
[15] : Wikipedia.org - Swinglabs, URL: <http://en.wikipedia.org/wiki/SwingLabs>
[16] : JavaFX Architecture, URL: <http://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm>
[17] : JavaFX Scene Graph, URL: <http://weblogs.java.net/blog/chet/archive/SceneGraph.pdf>
[18] : JavaFX Official Site, URL: <http://javafx.com/>
[19] : Guitar Fret Spacing, URL: <http://www.theguitarschool.com/Guitarfretspacing.html>
[20] : Wikipedia - Ladička, URL: <http://cs.wikipedia.org/wiki/Ladi%C4%8Dka>
[21] : Psychicorigami website - Guitar Tuner, URL: <http://www.psychicorigami.com/2009/01/17/a-5k-java-guitar-tuner/>
[22] : Wikipedia.org - Metronom , URL: <http://cs.wikipedia.org/wiki/Metronom>
[23] : MusicXML 3.0 Tutorial, URL: <http://downloads2.makemusic.com/MusicXML/musicxml-tutorial.pdf>

Přílohy

A Uživatelská příručka

Program slouží pro tvorbu hudební skladby a výuku hry na hudební nástroj. Poskytuje další doplňkové funkce.

A.1 Minimální požadavky

- CPU 1GHz
- 256 MB RAM
- běhové prostředí JRE, min 1.6

A.2 Ovládání aplikace

Při spuštění aplikace se zobrazí prázdné okno s hlavním menu. Pro práci a tvorbu s hudební skladbou je nutné vytvořit nový projekt, nebo otevřít uložený projekt.

A.2.1 Vytvoření nového projektu

Pro vytvoření nového projektu se zvolí položka v hlavním menu **File** -> **New Project**. Zobrazí se dialogové okno, ve kterém je možné si vybrat ze tří nabídek. Zvolením nabídky **Default project** se vytvoří základní struktura projektu, která obsahuje jednu notovou hudební stopu v houslovém klíči, jednu notovou hudební stopu v basovém klíči a jednu bicí hudební stopu. Zvolením nabídky **Empty project** se vytvoří prázdný projekt, do které se dají dále přidat nové hudební stopy. Nabídka **Custom project**, poskytne možnost vytvořit si vlastní nastavení hudebních stop.

Vytvořený projekt obsahující základní hudební stopy lze vidět na obrázku Obrázek A 2.1

Každá hudební stopa má svůj ovládací panel příklad na obrázku.

- Tlačítko **M** (Mute) – slouží k utlumení stopy

- Tlačítko **S** (Solo) – slouží k nastavené sólové stopy

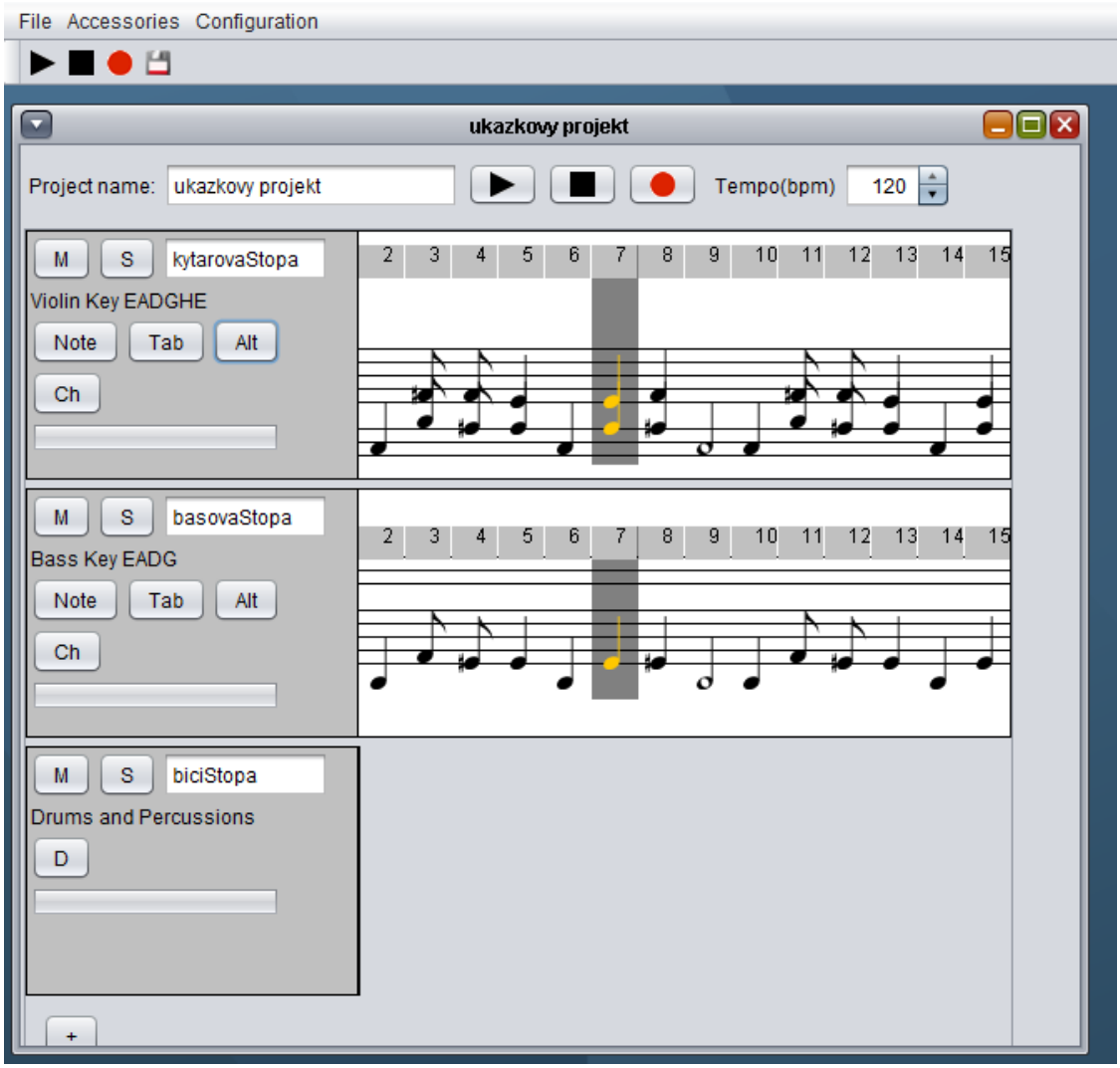
- Tlačítko **Note** – slouží k zobrazené notové osnovy

- Tlačítko **Tab** – slouží k zobrazení tabulaturové osnovy

- Tlačítko **Alt** – slouží k zobrazení alternativní osnovy

- Tlačítko **Ch** – slouží pro zobrazení nastavení kanálu stopy.

- Textové pole – obsahuje jméno stopy, které lze změnit



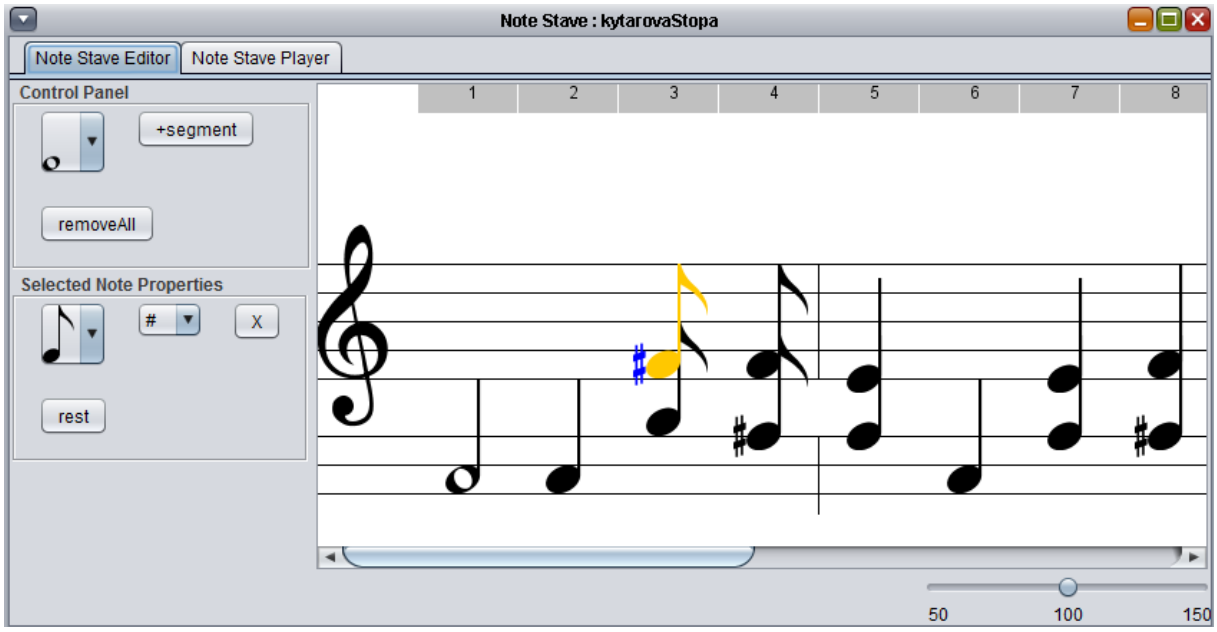
Obrázek A 2.1: Grafické uživatelské rozhranní aplikace se zobrazeným ukázkovým projektem

A.2.2 Ovládání hudebních stop

Zvolením tlačítka **Note** se zobrazí notová osnova určená pro editaci. Nachází se zde kontrolní panel, který obsahuje výběr tvaru nově vkládané noty, tlačítko **+segment**, sloužící pro přidání nových

segmentů pro zápis not a tlačítko **removeAll**, které smaže všechny noty v zápise. Nová nota se přidá zmáčknutím tlačítka myši v jednom ze segmentů zápisu na některé lince. Vložená nota bude mít tvar, který je nastaven v kontrolním panelu. Nota bude umístěna na lince, či mezi linkami, tam kde bylo kliknuto. Je možné noty přetahovat myší na jiné linky, či do jiných segmentů a to s omezenímy vyplývajícími z pravidel zápisu not, tedy například to, že v jednom segmentu musí mít všechny noty stejný tvar. Také to, že nemohou být dvě noty na stejném místě. Jakoukoliv notu je možné označit a změnit jí její vlastnosti. To se provede kliknutím na požadovanou notu. Následně se zobrazí panel pro změnu vlastností. Zde lze měnit tvar noty, ale také přidávat snížení, či zvýšení noty. Také lze nastvit tlačítkem **rest** notu na pomlku. Označenou notu je také možné tlačítkem **X** smazat z osnovy. Pomocí posuvnítka, lze měnit velikost zobrazené notové osnovy.

Lze se také přepnout na pohled na notovou osnovu pomocí záložky **Note Stave Player**, které zobrazuje vytvořený zápis a slouží pro zvýrazňování právě hraného tónu/akordu. Při vkládání not do notové osnovy se automaticky vytvářejí nové taby v tabulaturovém zápise. To funguje i naopak, kdy se vkládané taby automaticky převádějí na noty a jsou zobrazeny v notové osnově. Na obrázku



Obrázek A2.2: Ukázka notové editovatelné stopy

Zvolením tlačítka **Tab** se zobrazí tabulaturová osnova pro kytarový nástroj určená pro editaci. Opět se zde nachází kontrolní panel s výběrem tvaru noty nově vkládaného tabu, tlačítko pro přidání dalších segmentů **+segment** a také tlačítko **removeAll**, pro vymazání všech tabů z osnovy. Navíc je zde

kontrolní prvek, pomocí kterého je možné nastavit číslo pražce nově vkládaného tabu. Pro vložení nového tabu je opět nutní kliknout v některém segmentu na vybranou linku, která představuje kytarovou strunu. Vloží se tak tab s číslem pražce, které bylo zvoleno. Tab je také možné přetahovat myší na jiné linky a do jiných segmentů, podle pravidel zápisu tabů, kdy musí mít všechny taby v segmentu stejný tvaro noty. Dále není možné vkládat taby na stejné linky v jednom segmentu. Tab je možné označit a měni jeho číslo pražce, tvar noty, lze jej nastavit jako pomlku a lze jej také vymazat.

Zvolením tlačítka **Alt** se zobrazí alternativní zápis not. Alternativní noty jsou zobrazeny pomocí obrázků. Sadu těchto obrázků lze načíst pomocí konfiguračního souboru, pomocí nabídky **Configuration** -> **Load Alt Tones**.

Zvolením tlačítka **CH** – se zobrazí nastavení hudebního MIDI kanálu, kde lze vybrat z mnoha různých hudebních nástrojů, které hudební stopa má představovat.

A.2.3 Přehrávání hudebních stop

Vytvořenou hudební stopu lze přehrát tlačítkem **play**, které se nachází v nástrojovém panelu. Při přehrávání se zvýrazňují ve všech přehrávačích právě hrané tóny/akordy. Přehrávání lze zastavit tlačítkem **stop** v nástrojovém panelu. Změnu tempa skladby lze měnit pomocí ovládacího prvku **Tempo**.

A.2.4 Uložení a export

Celý projekt lze uložit do souboru v nabídce **File** -> **Save**, tím se vytvoří nový soubor s příponou ***.mol**.

Dále je možné projekt exportovat do různých formátů. V nabídce **File -> Export As..** jsou na výběr zvukový formát MIDI, vytvoří se tak nový soubor, který lze spustit jako samostatnou hudební skladbu pomocí nějakého hudebního přehrávače. Dále je možné exportovat notové hudební stopy do tzv. ASCII Tab, tedy tabulatury zapsané pomocí znaku ASCII. Nakonce je možné exportovat projekt do standartního formátu pro hudební notace **MusicXML**.

A.2.5 Načtení projektu ze souboru

Je možné uložený projekt v programu otevřít. To se provede v nabídce **File -> Open project**. Projekt je načten se všemi hudebními daty i nastavenímy pro všechny hudební stopy.

A.2.6 Doplňkové nástroje

Pro výuku slouží různé doplňkové nástroje v nabídce v hlavním menu **Accessories**. V položce **Virtual Instruments** se nacházejí dva virtuální hudební nástroje a to Piáno a Kytara. Při zobrazení okna virtuálního nástroje lze vybrat s kterou hudební stopou má být nástroj spojen. Lze tak syntetizovat hudbu interakvní prácí s nástrojem. Například na piánu lze mačkat jednotlivé klávesy. U kytary pak lze mačkat jednotlivé oblasti patřící pod určitou strunu a pražec zmáčknutý na této struně. Lze také označit po stisknutí tlačítka **marking** vybrané tóny na nástroji, které pak lze zahrát zároveň tlačítkem **push**.

Pomocí položky **Guitar Tuner** se zobrazí kytarová ladička, která slouží pro nalazení hudebního nástroje. Pro ladění je nutné mít mikrofon, nebo lze například elektrockou kytaru zapojit přímo do mikrofonního vstupu. Lazení se spustí tlačítkem **on**. Přibližný laděny tón nástroje se zobrazuje na grafickém rozhranní včetně odchylky od požadovaného.

Pomocí položky **Metronom** se zobrazí zařízení, které po spuštění tlačítkem **start** odklepává přesný rytmus v nastaveném tempu. Tlačítkem **stop** se odklepávání zastaví.

V nabídce **Configuration** -> **Look and Feel** lze zvolit dialog, vekterém je možné nastavit vzhled grafického rozhranní na jiné z předinstalovaných.

B. Příloha na DVD

V této příloze je zobrazena struktura přiloženého DVD, tedy obsažené soubory a adresáře:

- **NetBeans projekty implementací** – obsahuje soubory potřebné pro otevření implemnetací vytvořených programů pomocí vývojového prostředí NetBeans. Obsahuje další dva pod-adresáře:
  - MusicOnLine – implementace hlavního programu pro tvorbu a výuku hudby.
  - VirtualPianoFX – implementace doplňkového programu pomocí JavaFX
- **Spustitelné soubory** – obsahuje opět dva podadresáře:
  - MusicOnLine – obsahuje spustitelný souborový archiv **MusicOnLine.jar** a dávku pro spuštění apliakce **MusicOnline.bat**

- VirtualPianoFX – obsahuje spustitelný souborový archiv **VirtualPianoFX.jar**, JNLP soubor pro spuštění aplikace pomocí Java Web Start – **VirtualPianoFX.jnlp** a webovou stránku **VirtualPianoFX.html** pro spuštění aplikace skrze webový prohlížeč a pomocí Java Web Start

- **Ukázkový MusicOnLine projekt** – obsahuje soubor **ukazkovy projekt.mol**, který slouží jako demonstrativní ukázka funkce programu pro tvorbu hudby. Lze tedy otevřít v rámci programu MusicOnLine. V souboru je vytvořena hudební skladba skládající se z několika hudebních stop. Tu lze přehrát a dále s ní pracovat. V adresáři **export** jsou uloženy ukázkové exportované soubory programu. Soubor **alttonesdef.txt** slouží pro načtení alternativních tónů, uložených v adresáři **images** do projektu a zobrazování příslušné alternativní osnovy.
- **text.pdf –** PDF dokument, obsahující celou textovou část diplomové práce.